ABC de la mecatrónica

STEREN®

CONTENIDO

Capítulo 1

Mecatrónica

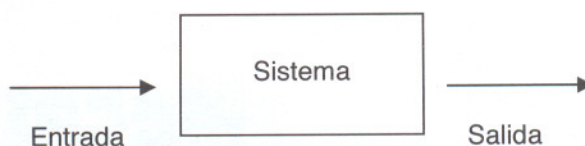
Mecatrónica

DEFINICIÓN

Aunque no es un concepto nuevo, este término ha adquirido una gran importancia en los últimos años por el impacto de sus aplicaciones. Por lo tanto, Mecatrónica no es una palabra simple de definir ya que se refiere a la automatización de procesos basada en la integración de los sistemas de control, concepto que analizaremos a lo largo del libro ya que el enfoque de la Mecatrónica considera a los sistemas como el núcleo de su análisis.

Mecatrónica se refiere al diseño integrado de los sistemas buscando un menor costo, una mayor eficiencia, una mayor confiabilidad y flexibilidad desde el punto de vista mecánico, eléctrico, electrónico, de programación y de control. La Mecatrónica adopta un enfoque integral desde estas disciplinas en lugar del enfoque secuencial tradicional del diseño partiendo de un sistema mecánico, luego el diseño de la parte eléctrica y luego su integración con un microprocesador.

La Mecatrónica se puede tomar como la oportunidad de analizar y resolver los problemas de automatización desde una perspectiva diferente e integral, donde los ingenieros no se deben limitar a considerar únicamente la solución desde el punto de vista de su especialidad, sino en el contexto de una gama de tecnologías. Este enfoque mecatrónico será conveniente para considerar el comportamiento de cada parte del sistema en función del resultado general esperado.

La Mecatrónica aborda su estudio partiendo del concepto de sistema. El sistema más simple puede considerarse como una estructura cerrada con una entrada y una salida en donde el principal interés es conocer la relación entre estas dos variables.

Por ejemplo, un termómetro podría considerarse un sistema de medición donde la entrada sería la magnitud que se quiere medir, es decir, la temperatura y, la salida sería el valor numérico registrado en el termómetro. Este sistema podría crecer y convertirse en un sistema de control de la temperatura, al cual podríamos llamar

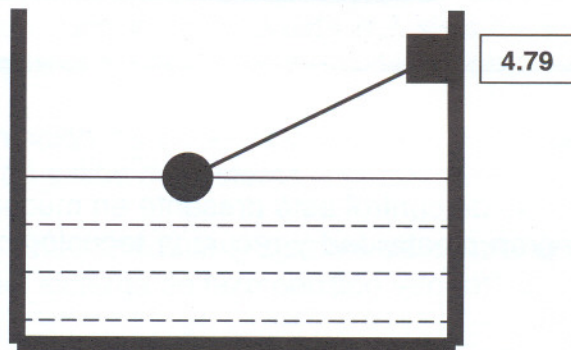
sistema de calefacción. De esta forma estaríamos hablando de un sistema de control, el cual fijaría una salida dentro de un rango programado.

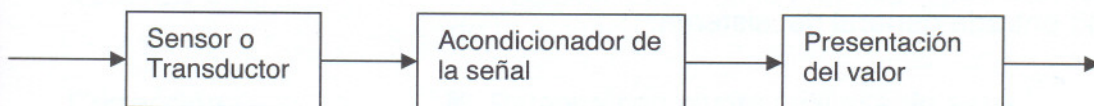
La Mecatrónica, en su parte de control, se auxilia de los desarrollos tecnológicos en sensores y transductores, sistemas de medición, actuadores, microprocesadores, microcontroladores y muchos más.

SISTEMAS DE MEDICIÓN

Revisemos brevemente dos de los sistemas básicos para el análisis y diseño de los proyectos mecatrónicos: los sistemas de medición y los sistemas de control.

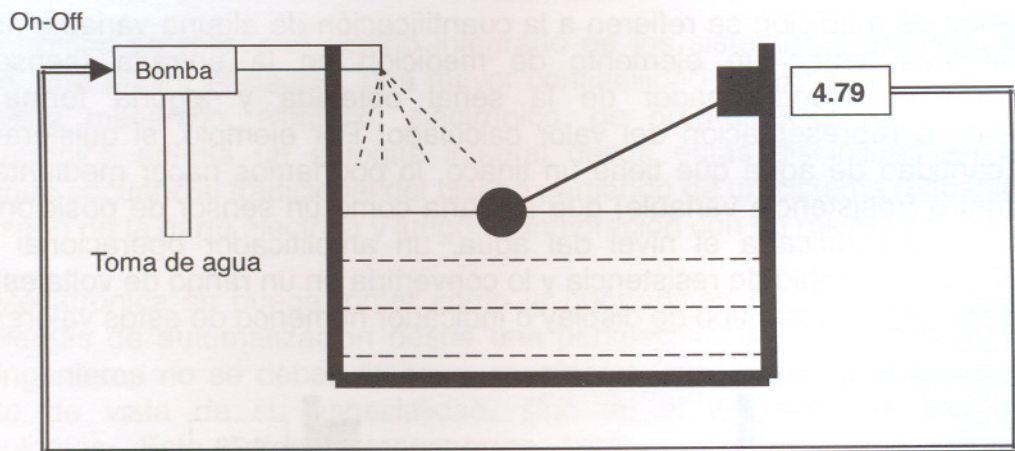
Los sistemas de medición se refieren a la cuantificación de alguna variable física. Estos sistemas tienen un elemento de medición en la entrada (sensor o transductor), un acondicionador de la señal obtenida y alguna forma de presentación o representación del valor calculado. Por ejemplo, si quisiéramos medir la cantidad de agua que tiene un tinaco, lo podríamos hacer mediante un potenciómetro (resistencia variable) que actuaría como un sensor de posición del flotador que nos indicaría el nivel del agua, un amplificador operacional que acondicionaría el cambio de resistencia y lo convertiría en un rango de voltajes y al final necesitaríamos algún tipo de display o indicador numérico de estos valores.

4.79

Su estructura básica representada en diagrama de bloques puede ser

→ Sensor o Transductor → Acondicionador de la señal → Presentación del valor →

SISTEMAS DE CONTROL

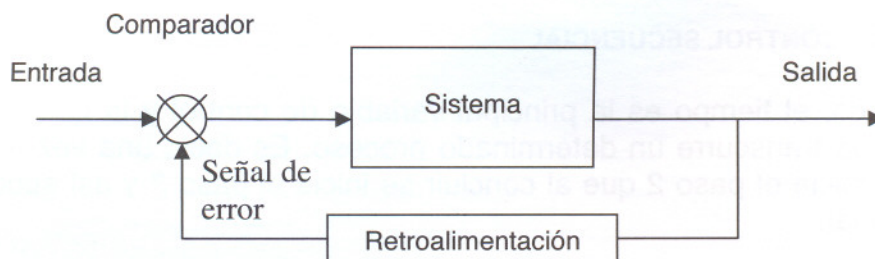
Por otro lado, los sistemas de control mantienen a lo largo del tiempo o de cualquier otro parámetro un valor constante o programado de la variable física que se está midiendo. Por lo tanto, los sistemas de control se basan en sistemas de medición, de los cuales es tomada su salida y retroalimentada como entrada al sistema.

Siguiendo con el ejemplo del tinaco, un sistema de control nos ayudaría a mantener el nivel del agua en un determinado punto o en un rango, según su programación.

On-Off
Bomba
4.79
Toma de agua

Esta idea de sistemas de control está presente en muchas actividades humanas y requieren de una gran creatividad y recursos tecnológicos para lograr soluciones de calidad.

SISTEMAS DE LAZO ABIERTO Y DE LAZO CERRADO

Los sistemas de control se pueden clasificar en sistemas de lazo abierto y en sistemas de lazo cerrado. Los sistemas de lazo abierto son sencillos y poseen poca capacidad de control. Los sistemas de lazo cerrado son más complejos y tienen control exacto sobre las variables de salida, ya que ésta es retroalimentada como entrada general del sistema.

Comparador
Entrada
Sistema
Salida
Señal de error
Retroalimentación

El comparador es el elemento que compara el valor programado como salida esperada (Entrada), contra el valor real de salida medido del sistema. Esta diferencia se considera como la señal de error.

Señal de error = valor programado como salida – valor real de salida

El bloque de sistema puede tener varios subsistemas internos que modifiquen de diferente forma la señal de entrada. Generalmente se podrá encontrar un subsistema de control que decida qué acción tomar sobre una señal de error, un subsistema de corrección que permita ejecutar una acción, a través de un actuador, para producir un cambio en la salida del sistema y un subsistema de proceso, que será la variable a controlar por parte del sistema.

El bloque de retroalimentación contiene el dispositivo de medición con un valor relacionado con la señal de salida del sistema.

Considerando el ejemplo anterior sobre el sistema de control de llenado del tinaco de agua, los elementos del sistema de lazo cerrado serían:

Salida	☒ Nivel del agua del tinaco
Entrada	☒ Posición del flotador
Comparador	☒ Palanca del flotador (potenciómetro)
Señal de error	☒ Posición inicial de la palanca – posición real de la palanca
Corrección	☒ Palanca con pivote / válvula de agua
Proceso	☒ Nivel de agua del tinaco
Dispositivo de medición	☒ Flotador y palanca

SISTEMAS DE CONTROL SECUENCIAL

En ocasiones, el tiempo es la principal variable de control y la que determina el orden en que transcurre un determinado proceso. Es decir, una vez terminado el paso 1, se inicia el paso 2 que al concluir se inicia el paso 3 y así sucesivamente hasta terminar.

En estos sistemas, el tiempo y la secuencia de eventos definen la estructura general de control. Desde luego que se puede hablar de un proceso secuencial de una sola variable o de un sistema de muchas variables.

Un ejemplo típico de sistema de control secuencial es la operación de una lavadora de ropa con un ciclo de lavado de 17 minutos, cuyo análisis se presenta en el siguiente diagrama.

Minutos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	Lavar								Enjuagar / Secar								
Llenar	X	X					X		X	X					X		
Agitar			X	X	X						X	X	X				
Centrifugar							X	X							X	X	X
Vaciar						X	X	X						X	X	X	X

Con esta gráfica de tiempo, analizamos la secuencia y la duración de cada uno de los cuatro procesos de una lavadora: llenar, agitar, centrifugar y vaciar.

Capítulo 2

Sensores y transductores

Sensores y transductores

DEFINICIÓN

Los sensores son dispositivos que miden la magnitud de una señal determinada y producen una señal relacionada. Los sensores usan las propiedades de los materiales de los que están hechos y así comparar su comportamiento ante las variaciones de la señal a medir. El termómetro de mercurio es un ejemplo en el que la sensible propiedad de dilatación que tiene el mercurio ante cambios de temperatura, se aprovecha para equipararla a su medición.

Muchos de los sensores más usados son eléctricos o electrónicos, aunque existen de otros tipos, y las magnitudes a medir son fenómenos físicos como diversos tipos de energía, velocidad, aceleración, tamaño, cantidad, etc. Como ejemplo podemos mencionar sensores de temperatura, humedad, fuerza, deformación, acidez, luz, sonido, contacto y proximidad.

Los transductores son dispositivos capaces de transformar un determinado tipo de energía de entrada en otra diferente y relacionada de energía de salida. Como ejemplo podemos mencionar transductores electroacústicos, electromecánicos, electromagnéticos, electroquímicos, fotoeléctricos, piezoeléctricos, termoeléctricos y de presión.

Los micrófonos, las bocinas, los teclados de los equipos y los ventiladores, son ejemplos prácticos de aplicaciones de transductores.

CARACTERÍSTICAS DE OPERACIÓN

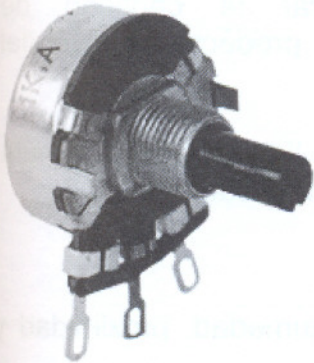
El funcionamiento y evaluación de los transductores se basa en diferentes características de operación, las cuales se pueden ampliar y convertirse en las características de los sistemas de medición en su conjunto. Independientemente del tipo de transductor del que se trate, éstas siempre estarán presentes.

Las características más consideradas son:

- Intervalo. Es el rango en magnitud que puede tener la señal de entrada
- Extensión. Es el valor máximo de entrada que puede detectar un transductor
- Resolución. Es la mínima señal de cambio en la señal de entrada detectada por el transductor
- Sensibilidad. Es la relación que existe entre la entrada y la salida del transductor

- Error. Es la variación existente entre el valor real de la señal y el valor registrado por el transductor
- Exactitud. Es la capacidad de reproducir la misma señal de salida a la misma señal real de entrada suponiendo un error constante del transductor
- Histéresis. Es la exactitud en la señal de salida considerando si los cambios en la señal de entrada son por incrementos o por decrementos de valor.
- Linealidad. Es la exactitud que se obtiene en el intervalo de operación del transductor
- Estabilidad. Es la garantía de exactitud durante el mayor periodo de tiempo de uso del transductor.
- Acoplamiento. Se refiere a la impedancia de salida del transductor que afecta el circuito en el que se conecta.

SENSORES RESISTIVOS

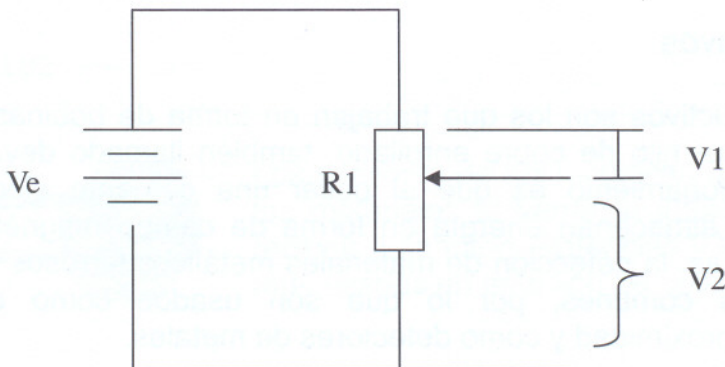
Los sensores resistivos son los que aprovechan los cambios de resistencia en su material para medir la señal asociada. Este tipo de sensores basados en la variación de la resistencia eléctrica son muy usados ya que son muchas propiedades físicas que la afectan, pudiendo clasificarse en mecánicas, térmicas, ópticas y químicas.

Un potenciómetro es una resistencia variable cuyo valor se determina por el desplazamiento de un contacto móvil deslizante o giratorio. Este desplazamiento, se convierte en una diferencia de potencial, de donde se vuelve un sensor muy usado.

El diagrama eléctrico equivalente es muy similar al de la resistencia, solamente que tiene un indicador del contacto móvil, llamado también cursor.

Ve

R1

V1

V2

Estas diferencias de potencial (V1 – V2) se pueden asociar a un modelo de dos resistencias en serie.

Una aplicación de sensores resistivos son los detectores de temperatura basados en la variación de resistencia eléctrica de sus materiales y son denominados como RTD (Resistance Temperature Detectors). La resistencia nominal de un termistor se elige fundamentalmente con base al alcance de temperatura de operación. Mayores valores de resistencia corresponden a temperaturas más elevadas, mientras las bajas temperaturas requieren menores resistencias.

SENSORES CAPACITIVOS

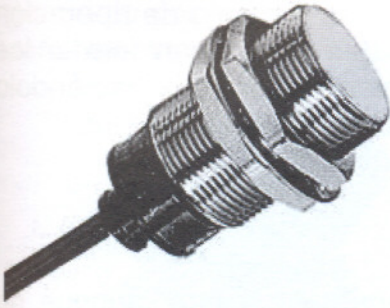
Los sensores capacitivos son sensores que sus materiales se comportan como condensadores eléctricos. Los condensadores son dispositivos que, sometidos a una diferencia de potencial (voltaje) adquieren una determinada carga eléctrica. A esta propiedad de almacenamiento de carga se le denomina capacidad. Los condensadores están formados de dos placas o láminas conductoras separadas por un material dieléctrico.

Una aplicación de sensores capacitivos son los detectores de nivel, los cuales modifican sus características al modificar la cantidad de dieléctrico entre sus placas, producto del nivel del líquido que están midiendo.

En ocasiones es el mismo líquido el que simula ser la segunda placa del condensador.

Este tipo de sensores también son usados para medir humedad, proximidad y posición.

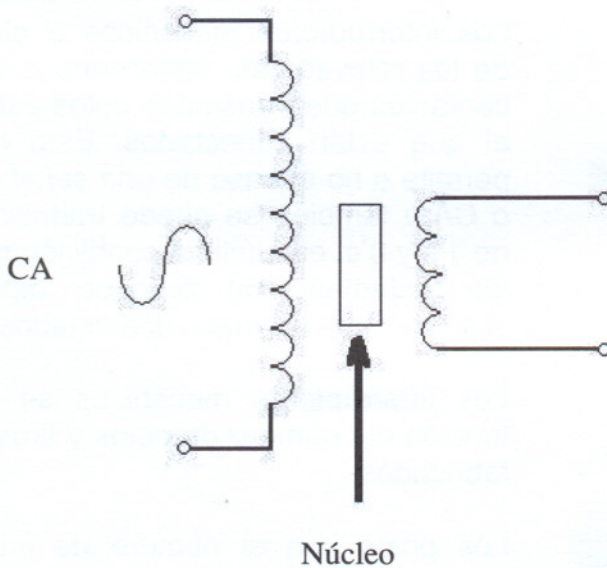
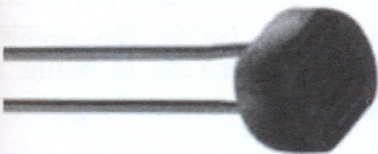
SENSORES INDUCTIVOS

Los sensores inductivos son los que trabajan en forma de bobinas. Las bobinas están formadas por hilo de cobre enrollado, también llamado devanado, y cuyo principio de funcionamiento es que al pasar una corriente eléctrica por sus terminales, éstas almacenan energía en forma de campo magnético. Debido a estas características, la detección de materiales metálicos ferrosos es una de sus aplicaciones más comunes, por lo que son usados como detectores de posicionamiento, proximidad y como detectores de metales.

La inductancia de las bobinas depende del diámetro del cable del que están hechas y del número de vueltas de su fabricación.

Al igual que los sensores capacitivos, son sensores que responden muy bien en sistemas de corriente alterna, siendo la frecuencia de la señal que los estimula una de las variables a medir para identificar sus cambios.

Esta capacidad de almacenamiento de energía magnética de las bobinas, es usada para afectar a otras bobinas. Es así como se forman los transformadores. Los transformadores diferenciales de variación lineal (LVDT) son los que modifican el campo magnético entre las bobinas al modificar la posición de su núcleo.

CA

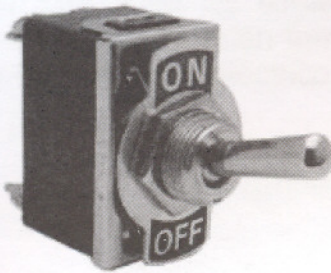
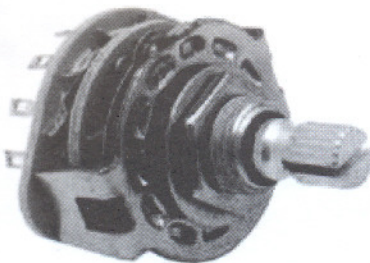
Núcleo

SENSORES DE LUZ

Los sensores de luz modifican las propiedades de los materiales al variar la intensidad de luz que reciben. Las fotorresistencias y los fotodiodos son muy usados en este tipo de aplicaciones.

Los fotodiodos tienen una mejor respuesta lineal que las fotorresistencias que, en cambio, ofrecen un alto valor de resistencia en la oscuridad pudiendo llegar a 2 MOhms, y un valor de resistencia de 50 KOhms en mayor iluminación.

Los fotodiodos son semiconductores, por lo que basan su principio de operación en el comportamiento de las uniones P-N. De aquí que se puedan formar los fototransistores, en los que la luz incide sobre la región de la base, haciéndolo más sensible que el fotodiodo por el efecto de ganancia propio del transistor.

INTERRUPTORES

Los interruptores pueden ser un tipo especial de sensores ya que, aunque no modifican las propiedades físicas de sus componentes, sirven para detectar diferentes estados de fenómenos u objetos que se quieran medir.

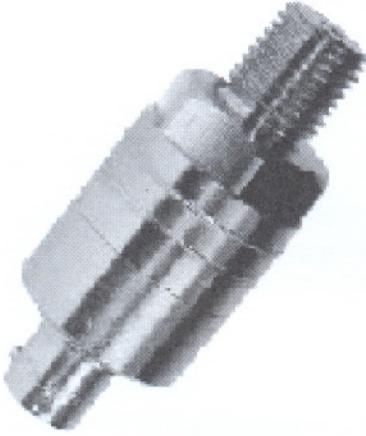
La construcción de los interruptores ofrece una gama de posibilidades para detectar movimientos, posiciones y frecuencias de comportamiento de estos objetos.

Los interruptores mecánicos o electromecánicos de los relevadores, tienen uno o varios pares de contactos que transmiten estos estados al circuito al que están conectados. Este comportamiento permite o no el paso de una señal eléctrica de CC o CA y también se puede traducir como señales de 1's y 0's, esta última condición muy utilizada en los sistemas con enfoque digital, es decir, sistemas que permiten dos estados.

Los interruptores mecánicos se especifican en función del número de polos y tiros con que están fabricados.

Los polos son el número de interruptores que funcionan a la vez en cada posición y los tiros son el número de contactos por posición.

En aplicaciones electrónicas, uno de los problemas que se presenta con el uso de interruptores mecánicos es el rebote físico de sus contactos. Este rebote puede ser interpretado por la electrónica del circuito en el que está conectado como una señal diferente, por lo que hay que tomarlo en cuenta y hacer los ajustes necesarios. Algunas medidas pueden ser el retardo en la lectura de la señal, el uso de circuitos de memoria adicionales o el uso de circuitos tipo *schmitt triggers*.

TRANSDUCTORES DE PRESIÓN

Los nombres de los transductores indican la transformación y medición que hacen. De esta forma, cualquier dispositivo que convierta el movimiento mecánico

generado por fuerzas asociadas a una presión externa y que se traduzca en una señal eléctrica o electrónica, se podrá considerar como un transductor de presión.

Estos pueden tener diferentes encapsulados, según la aplicación para la que están destinados.

La presión se puede definir como la fuerza que se ejerce sobre una unidad de área conocida y muchas veces se mide en comparación a otra columna de referencia. Los transductores lo harán midiendo el desplazamiento o deformación de su membrana o diafragma.

La presión también puede ser producto del volumen de un fluido (líquido o gas).

Acondicionamiento de señales analógicas

DEFINICIÓN

Las señales obtenidas de los sensores y transductores que se usan en los sistemas de medición, tienen que ser procesadas y adaptadas para poder pasarlas a la siguiente etapa.

Este proceso de adaptación es lo que se conoce como acondicionamiento de la señal y como ejemplo de estos cambios se pueden mencionar los siguientes:

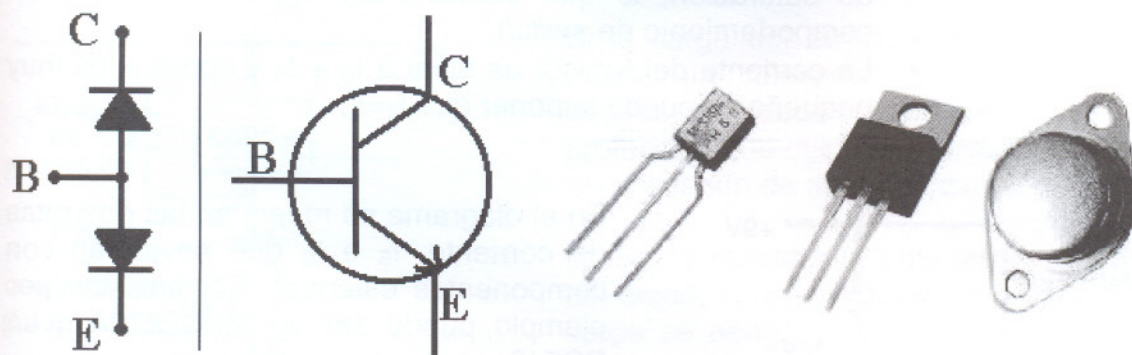
- La señal del sensor o transductor es demasiada pequeña, por lo que hay que amplificarla para que se pueda acoplar en la siguiente etapa
- La señal tiene interferencias no deseadas, lo cual puede ser muy común cuando los transductores no tienen un buen aislamiento eléctrico
- La señal del transductor es de tipo analógico y hay que acoplarla a un sistema digital, por lo que habrá que pasarla por un conversor analógico-digital
- La salida del transductor no tiene la impedancia adecuada para la siguiente etapa, lo que causaría pérdida de la señal de salida
- La señal del transductor es un voltaje de DC y hay que cambiarlo a pulsos
- En las primeras etapas de los sistemas de instrumentación es común que se tengan que acoplar y adaptar señales de corriente y de voltaje que sean proporcionales
- La señal de salida del transductor no es lineal, por lo que es necesario hacer una corrección

Este acondicionamiento de las señales de los transductores puede ser simple o complejo y hay elementos y configuraciones estándar para su tratamiento.

EL TRANSISTOR

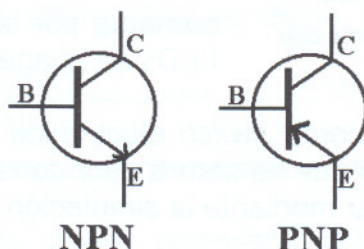
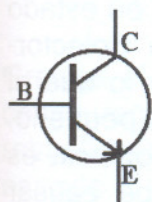
Los transistores son dispositivos semiconductores que debido a sus características de operación pueden desarrollar varias funciones principales. Una es de amplificador de señales, otra es de ser un circuito de conmutación, es decir, un interruptor (*switch*) electrónico, de oscilador y también puede ser un rectificador de señales.

Los transistores están formados por tres capas semiconductoras que forman dos uniones bipolares y que pueden tener un orden P-N-P o N-P-N, por lo que en el diseño de circuitos son considerados como elementos activos, a diferencia de los elementos pasivos formados por condensadores, resistencias y bobinas.

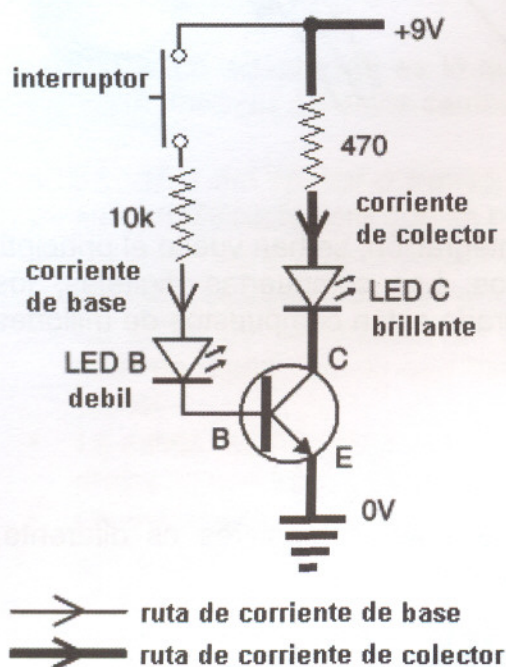
Cada capa tiene su nombre, por lo que el transistor tiene un colector (c), un emisor (E) y una base (B). Los transistores pueden tener diferentes encapsulados y su representación para el diseño de circuitos se muestra a continuación.

Los transistores, gracias a las tecnologías de integración, se han vuelto el principal componente de todos los circuitos integrados. Las compuertas digitales, los microprocesadores y todo tipo de circuito integrado están compuestos de millones de transistores microscópicos.

POLARIZACIÓN DEL TRANSDUCTOR

Como la polarización de los dos tipos de transistores bipolares es diferente, empezaremos por diferenciar su diagrama eléctrico.

NPN PNP

Independiente a que la elaboración de los transistores NPN es más simple, su uso puede estar definido por los criterios de diseño del circuito, ya que usándolo como *switch*, si el estímulo es un 1 convendrá usar un NPN y si el estímulo es un 0 entonces la elección será un PNP.

La operación de un transistor se basa en los siguientes conceptos

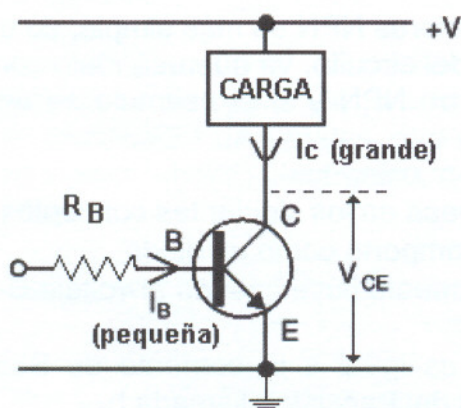
- La unión Base-Emisor se comporta como un diodo
- Una corriente de Base solamente fluye cuando el voltaje B-E es mayor a 0.7 Volts
- La corriente del Colector es igual a la corriente de Base aumentada por la ganancia del transistor, llamada h_{FE}

- Se necesita incluir una resistencia en serie con la Base del transistor para limitar su corriente y evitar dañarlo
- En un extremo de operación, el transistor entraría en estado de saturación, lo que causaría un V_{CE} de 0 volts y un comportamiento de *switch*
- La corriente del Emisor es igual a $I_C + I_B$ y como I_B es muy pequeña se puede suponer que $I_E=I_C$

En el diagrama se muestran las dos rutas de corriente, I_B e I_C que se logran con componentes estándar. El transistor, por ejemplo,.puede ser un MPS2222A o un BC548.

La corriente de Base es la que controla la corriente del Colector.

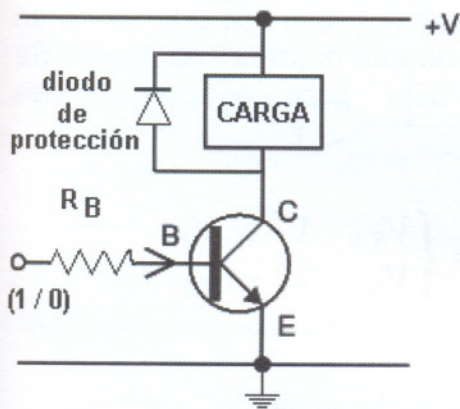
Cuando el interruptor se cierra, la pequeña corriente de Base fluye y prende el LED B con una intensidad débil. El transistor amplifica la corriente que fluye del Colector al Emisor, por lo que el LED C prende con una intensidad brillante.

Cuando el interruptor se abre, no fluye corriente por la Base, por lo que los dos LEDs se apagan.

Cuando el transistor se usa como *switch* electrónico la polarización se va a los extremos de operación, por lo que se satura para cerrarlo y se apaga para abrirlo. El dispositivo a prender/apagar mediante la simulación de abrir/cerrar del *switch* se le llama carga.

La forma de operación en un transistor en configuración de *switch* es:

- Cuando el transistor está apagado la corriente de colector es cero, por lo que $I_C \times V_{CE} = 0$
- Cuando el transistor está en estado de saturación, el voltaje Colector-Emisor es casi cero, por lo que el producto $I_C \times V_{CE}$ es muy pequeño, lo que significa que la potencia es baja y entonces no debe causar calentamiento en el transistor.

Los valores importantes a considerar es esta configuración son la máxima corriente de colector y la mínima ganancia de corriente h_{FE}.

diodo de protección, CARGA, +V, R_B, B, C, E, (1 / 0)

Si la carga del *switch* electrónico es un motor, un relevador o algún tipo de bobina, se debe incluir un diodo de protección que cuide al transistor de algún daño cuando de apaga la carga.

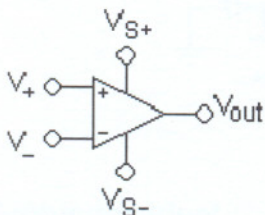
La figura muestra la polarización inversa del diodo, lo que garantiza que cuando la carga se apaga, la corriente fluya por el diodo y protege al transistor.

El transistor es un buen *switch* para DC, por lo que si lo que si se quiere controlar corriente AC o alto voltaje, se tendrá que usar un relevador.

AMPLIFICADORES OPERACIONALES

Los amplificadores operacionales son amplificadores de señal con una configuración general prediseñada y que se encuentra encapsulada en un circuito integrado. Tres de las características importantes son:

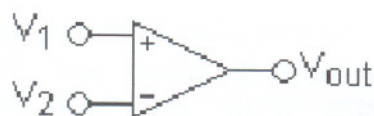
- Tienen una entrada diferencial, es decir, una entrada positiva y una negativa
- La salida del amplificador se define como $V_{OUT} = G\ (V_+ - V_-)$, por lo que la ganancia es $g = Vo / Vi$
- En forma ideal, la impedancia de entrada en infinita y la de salida es nula

V_{S+}, V_+, V_-, V_{out}, V_{S-}

Esta figura representa el diagrama eléctrico de los amplificadores operacionales.

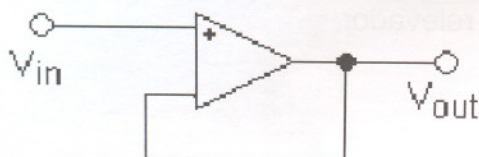
Los voltajes V_S son voltajes de polarización que, generalmente, pueden llegar a los niveles de 15 Volts.

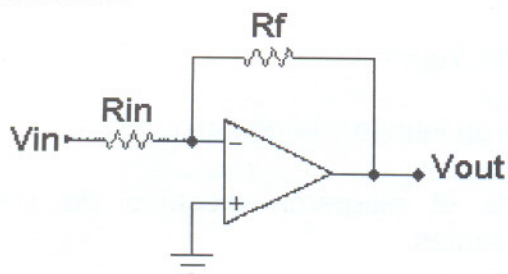
El modelo uA741 de la compañía Fairchaild es uno de los más conocidos.

Gracias al diseño de los amplificadores operacionales, estos pueden tener una configuración de lazo abierto y también de lazo cerrado. Si el voltaje de salida se retroalimenta como voltaje de entrada negativo, la configuración será de lazo cerrado. Los amplificadores operacionales pueden tener diferentes configuraciones, lo que los hace un dispositivo muy atractivo para acondicionar señales.

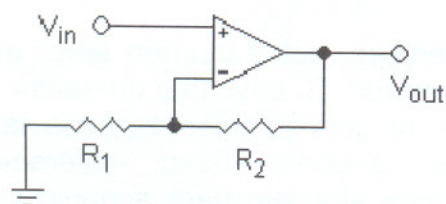
Comparador

- Esta es una aplicación sin la retroalimentación. Compara entre las dos entradas y entrega una salida en función de qué entrada sea mayor. Se puede usar para adaptar niveles lógicos

$$V_{out} = \begin{cases} V_{S+} & V_1 > V_2 \\ V_{S-} & V_1 < V_2 \end{cases}$$

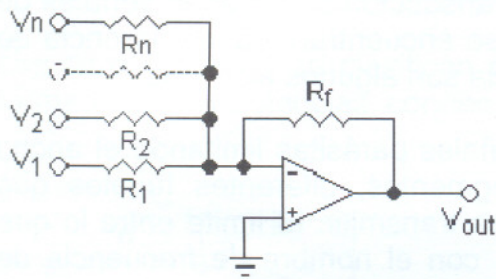
Seguidor

- El seguidor es un circuito que proporciona a la salida un voltaje igual que a la entrada
- Se usa como un *buffer*, para eliminar efectos de carga o para adaptar impedancias (conectar un dispositivo con gran impedancia a otro con baja impedancia y viceversa)

Inversor

- El inversor es un amplificador que cambia el signo o la fase del voltaje de entra. La ganancia está determinada por la relación de resistencias

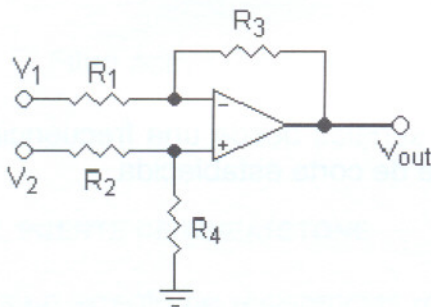
$$V_{OUT} = -V_{in}\frac{R_f}{R_{in}}$$

No Inversor

- El no inversor es un amplificador de señal con una ganancia determinada por la relación de resistencias.

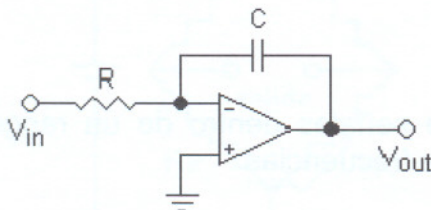
$$V_{out} = V_{in}(1 + \frac{R_2}{R_1})$$

Sumador Inversor

- El amplificador suma los voltajes de entrada e invierte la señal de salida

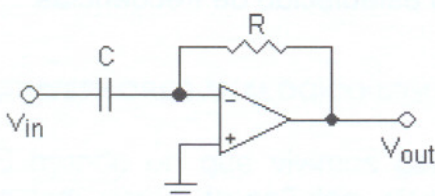
$$V_{out} = -R_f\left(\frac{V_1}{R_1} + \frac{V_2}{R_2} + ... + \frac{V_n}{R_n}\right)$$

Restador (amplificador diferencial)

- Esta configuración resta los voltajes de entrada. La polaridad del voltaje de salida depende de la magnitud de los valores de entrada

$$V_{out} = V_2\left(\frac{(R_3 + R_1)\,R_4}{(R_4 + R_2)\,R_1}\right) - V_1\left(\frac{R_3}{R_1}\right)$$

Integrador Ideal

- La señal de entrada se integra como una función dependiente del tiempo
- La señal de salida está invertida
- $V_{inicial}$ es el voltaje cuando t=0

$$V_{out} = \int_0^t -\frac{V_{in}}{RC}\,dt + V_{inicial}$$

Derivador ideal

- La señal de entrada se deriva y se invierte con respecto al tiempo

$$V_{out} = -RC\,\frac{dV_{in}}{dt}$$

FILTROS ANALÓGICOS

Algunas señales que entregan los sensores y transductores contienen señales de interferencia, producto del ambiente en el que se encuentran. La interferencia de la línea de voltaje o de señales de radiofrecuencia son algunos ejemplos.

Los filtros analógicos pueden eliminar estas señales parásitas limitando el ancho de banda a través de generar diferentes segmentos, diferentes túneles que permitan pasar solamente la señal que se desea transmitir. El límite entre lo que se pasa y entre lo que se rechaza se conoce con el nombre de frecuencia de corte.

Los filtros se clasifican de acuerdo con los segmentos de frecuencia que dejan pasar o que rechazan. De esta forma la clasificación puede ser de cuatro tipos diferentes:

1. Filtro pasa bajas

   Permite el paso de señales desde una frecuencia 0 hasta la frecuencia de corte establecida

2. Filtro pasa altas

   Permite el paso de señales a partir de la frecuencia de corte establecida

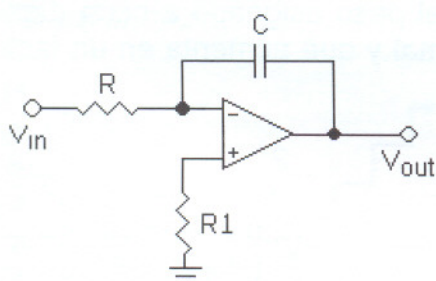
3. Filtro pasa banda

   Permite el paso de señales dentro de un rango superior e inferior de frecuencias

4. Filtro supresor de banda

   Permite el paso de señales en todo el espectro excepto en un rango establecido de frecuencias

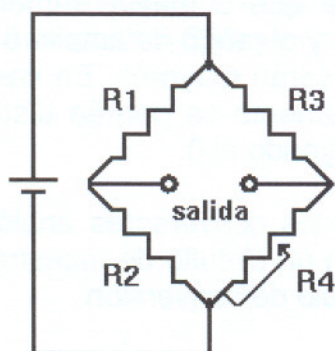
En los filtros, la frecuencia de corte se considera cuando la señal alcanza el 70.7% de su valor, lo que equivale a una atenuación de 3 dB.

Los filtros también se clasifican en pasivos y activos. Los filtros pasivos están formados por resistencias, condensadores y bobinas.

Filtro activo pasa bajas

Una de sus desventajas es que la frecuencia se puede modificar por el consumo de energía de los componentes.

Los filtros activos se refieren a los circuitos con elementos semiconductores como transistores y amplificadores operacionales y no tienen la desventaja de los filtros pasivos. Las configuraciones de Integrador y Derivador de los amplificadores operacionales son usadas como filtros.

EL PUENTE DE WHEATSTONE

Es un arreglo de resistencias muy usado para medir los cambios en una de ellas, lo cual produce un cambio de voltaje en su salida.

Este circuito resistivo puede operar con voltaje directo en un rango menor a 12 volts.

Cuando el puente está en equilibrio, significa que el voltaje de salida es 0, por lo que implica que R1=R2 y R3=R4.

Si la resistencia variable R4 es el elemento sensor que se está usando y cambia su valor, el voltaje de salida va a cambiar en consecuencia.

En ocasiones, estos sensores requieren de un elemento de compensación para evitar cambios adicionales como podría ser por factores de temperatura.

CONVERSORES ANALÓGICO-DIGITALES

El mundo en que vivimos genera señales de tipo analógicas. Los sensores y transductores de señales, siempre van a generar señales analógicas. Para que

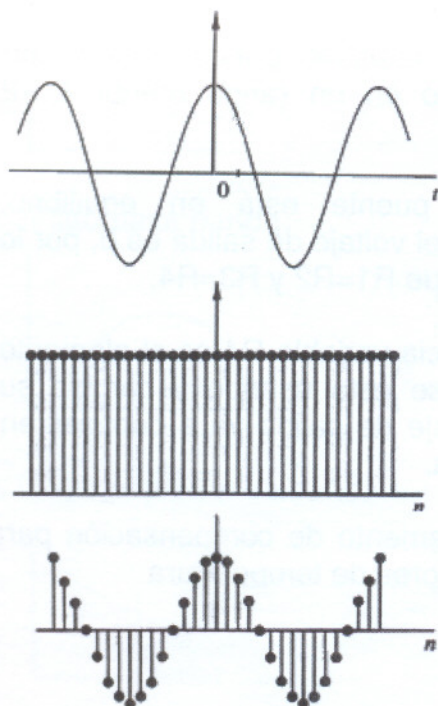
estas señales puedan ser incorporadas a sistemas digitales del tipo circuitos electrónicos digitales o computadoras, es necesario cambiar estas señales analógicas a señales digitales.

Para manejar esta conversión, el sistema binario, con dígitos 0 y 1, es la base teórica para poder manejarla. Estos dígitos binarios, desde el punto de vista de la electrónica, son llamados bits. Cuando un número se representa por este sistema, la posición del dígito en el número binario indica el peso asignado a cada dígito, peso que tiene un equivalente en un sistema decimal y que aumenta en un factor de 2, representado por la expresión 2^n

2^3	2^2	2^1	2^0
8	4	2	1
Bit 3	Bit2	Bit1	Bit0

Como ejemplo, el número decimal 12 en un sistema binario tiene la siguiente representación

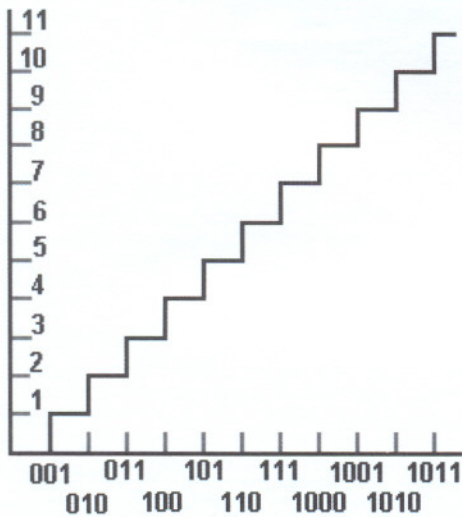
2^3	2^2	2^1	2^0
8	4	2	1
1	1	0	0

La comprobación de 8 + 4 confirma que la conversión fue correcta.

Según el número de bits, será la capacidad del conversor, ya que a mayor número de bits, la resolución y el rango de amplitud de la señal a manejar serán mayores. En caso de usar 4 bits, solamente se podrán distinguir 16 números incluyendo el 0.

La operación de los conversores analógico-digital se basa en un circuito de muestreo de la señal y el módulo de conversión.

En la parte superior de la figura se ve la señal que se quiere convertir. En la parte central se representa el circuito de muestreo que tomará una lectura de la señal con una frecuencia fija. En la parte inferior se ven los valores recolectados por el circuito de muestreo que simula la señal original, mostrando solamente los valores muestreados.

Estas muestras se pasan al conversor analógico-digital y así se obtendrá su equivalente en el número de bits que tenga el conversor.

La frecuencia de muestreo de una señal es importante, ya que define la fidelidad con la que se va a reproducir en forma binaria. El teorema de Nyquist - Shannon, conocido como el teorema de muestreo, dice que para poder ser reconstruida, la frecuencia de muestreo debe ser por lo menos del doble de la frecuencia máxima de la señal original.

Hay varios métodos de conversión analógico-digital. El método de escalera es muy usado en dispositivos de bajo precio y consiste en incrementar la cuenta binaria hasta que ésta coincida con el valor analógico a comparar.

La figura ilustra una entrada analógica en el eje vertical y una salida digital equivalente en el eje horizontal.

La longitud de la palabra binaria determina la resolución del conversor, definiendo el cambio de señal más pequeño que puede detectar a su salida.

De esta manera un convertidor analógico-digital de una longitud de palabra de 8 bits, con un intervalo de señal analógica de 5 volts, el número de niveles en una palabra es de $2^8 = 256$, y por ello la resolución es 5/256 = 19.5 mV.

Capítulo 4

Electrónica digital

Electrónica digital

DEFINICIÓN

La electrónica digital ha alcanzado un excelente nivel de integración y en la actualidad un sinnúmero de dispositivos basan su operación en estas tecnologías. Igualmente sucede con los sistemas de control que se aplican a diversos mecanismos. Una característica importante de estos sistemas es que dependen de dos únicos valores de operación, el "0" y el "1", generalmente asociados a voltajes de 0 y +5 volts.

La teoría que rige la operación de estos circuitos es el álgebra de Boole que tiene su equivalente en compuertas lógicas. El término lógica combinacional se refiere a la combinación de dos o más compuertas lógicas para ofrecer el resultado deseado.

SISTEMAS NUMÉRICOS

Los fenómenos físicos y químicos siempre tienen una representación en valores decimales. El sistema decimal se representa a partir de 10 dígitos diferentes: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. En cualquier sistema numérico, la posición del dígito indica el peso asignado el cual, en el sistema decimal, aumenta en un factor de 10 en cada posición que aumenta a la izquierda.

....	10^3	10^2	10^1	10^0
	Millares	Centenas	Decenas	Unidades

Así se puede mostrar la equivalencia de los números decimales, como por ejemplo el 2,575.

10^3	10^2	10^1	10^0
2	5	7	5

El sistema binario se representa a partir de 2 dígitos diferentes: 0 y 1. Igualmente, la posición del dígito indica su peso asignado el cual aumenta en un factor de 2 en cada posición que aumenta a la izquierda. Estos dígitos binarios se les denominan bits.

....	2^3	2^2	2^1	2^0
	Bit3	Bit2	Bit1	Bit0

Así se puede mostrar la equivalencia al número decimal 13

2^3	2^2	2^1	2^0
1	1	0	1

Agrupando de tres en tres los dígitos binarios de derecha a izquierda, se logra una conversión al sistema octal. El sistema octal se representa por 8 dígitos diferentes: 0, 1, 2, 3, 4, 5, 6 y 7. La posición del dígito indica su peso asignado el cual aumenta en un factor de 8 en cada posición que aumenta a la izquierda.

.... 8^3 8^2 8^1 8^0

Así se puede mostrar la equivalencia al número binario 111001100, 460 decimal.

8^2	8^1	8^0
7	1	4

Agrupando de cuatro en cuatro los dígitos binarios de derecha a izquierda, se logra una conversión al sistema hexadecimal. El sistema hexadecimal se representa por 16 dígitos diferentes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. La posición del dígito indica su peso asignado el cual aumenta en un factor de 16 en cada posición que aumenta a la izquierda.

.... 16^3 16^2 16^1 16^0

Así se puede mostrar la equivalencia al número binario 100110101100, 2,476 decimal.

16^2	16^1	16^0
9	A	C

En cualquier sistema numérico excepto en el sistema binario, los enteros positivos se representan sin signo y los enteros negativos se representan con un signo de menos al principio. Como el sistema binario es usado en las computadoras, hay una limitación impuesta por el hardware en la que todo debe representarse en formato binario. Para tal efecto, los números negativos se representan con el bit más significativo de la palabra que se usa. La convención es que el bit es cero si el número es positivo y uno si es negativo. Normalmente no hay problema para identificar los bits si se define con anticipación el tipo de representación a usar.

En electrónica se usa un equivalente adicional, el sistema decimal de codificación binaria (BCD) muy usado también por las computadoras. Este sistema representa los números decimales en forma binaria pero en forma independiente, no como una cantidad única. Así, el número 23 decimal se representa como 010 011 en formato BCD.

Este código es útil para las salidas de los sistemas basados en microprocesadores y que ofrecen salida a *displays* numéricos.

La siguiente tabla representa un equivalente entre los sistemas numéricos.

Decimal	Binario	Octal	Hexadecimal	BCD
0	0000	0	0	0000 0000
1	0001	1	1	0000 0001
2	0010	2	2	0000 0010
3	0011	3	3	0000 0011
4	0100	4	4	0000 0100
5	0101	5	5	0000 0101
6	0110	6	6	0000 0110
7	0111	7	7	0000 0111
8	1000	10	8	0000 1000
9	1001	11	9	0000 1001
10	1010	12	A	0001 0000
11	1011	13	B	0001 0001
12	1100	14	C	0001 0010
13	1101	15	D	0001 0011
14	1110	16	E	0001 0100
15	1111	17	F	0001 0101

CÓDIGO DE CARACTERES ASCII

La manipulación de los números por las computadoras se puede extender a la manipulación de letras. Por lo tanto, además de tener una representación y manipulación de números, es necesario tener una representación de letras y de caracteres adicionales usados en los textos.

El código binario estándar para los caracteres alfanuméricos es el llamado código ASCII (*American Standard Code for Information Interchange*). Este código usa una longitud de palabra de 7 bits, lo que permite representar 128 caracteres diferentes. Estos incluyen letras minúsculas, mayúsculas y caracteres de puntuación. Los primeros 33 caracteres (incluyendo el 0) y el 128 son de control, por lo que no son caracteres imprimibles.

El código ASCII se representa con siete bits, pero las computadoras manejan una palabra mínima de ocho bits, llamada byte. Por esta razón, lo más común es almacenar el código ASCII en ocho bits Este bit extra se utiliza para otros fines, que generalmente se usa para dar indicaciones adicionales a la impresora, ya que este código se usa para tales fines.

A continuación se muestra una tabla ASCII codificada en siete bits, donde se muestran los caracteres de control, el abecedario en minúscula y mayúscula y los caracteres especiales.

	0	16	32	48	64	80	96	112
0	NULL	DLE	sp	0	@	P	‘	p
1	SOH	DC1	¡	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ANQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
10	LF	SUB	*	:	J	Z	j	z
11	VT	ESC	+	;	K	[	k	{
12	FF	FS	,	<	L	\	l	|
13	CR	GS	-	=	M	]	m	}
14	SO	RS	.	>	N	^	n	~
15	SI	US	/	¿	O	_	o	DEL

OPERACIONES ARITMÉTICAS

Las operaciones aritméticas en los diferentes sistemas numéricos siguen las reglas de la aritmética ordinaria. Si los signos son iguales, las magnitudes se operan y guardan el mismo signo. Si los signos cambian, los resultados cambiarán en consecuencia.

En el sistema binario las operaciones con números negativos usan una codificación diferente, llamada sistema de complemento con signo. Mientras que el sistema de magnitudes usa el bit más significativo a uno para indicar que el signo es negativo, en este nuevo formato los números negativos se indican en su complemento a uno o dos.

Por ejemplo, considerando una representación de 8 bits el número +10 se representa en forma binaria como 00001010, el -10 suponiendo el sistema de magnitud con signo como 10001010, en complemento a uno como 11110101 y en complemento a dos como 11110110.

La suma de dos números binarios con signo positivo, se regirá por las reglas ordinarias. La suma entre números binarios positivos y negativos, se convertirá el número negativo a formato de complemento a dos y se resolverá sumando los dos operandos. El resultado quedará en complemento a dos, por lo que habrá que hacer otra conversión para tener la magnitud real.

Para representar el comentario anterior, se presenta la suma 8 + 10 y 8 – 10.

8	00001000	8	00001000
+ 10	+ 00001010	- 10	- 11110110
+ 18	00010010	- 2	11111110

La resta de dos números binarios con signo negativo y que están en formato de complemento a dos, se resuelve en una forma sencilla: se obtiene el complemento a dos del sustraendo y se suma al minuendo. En caso de un acarreo en la operación, éste se desprecia. Como ejemplo, la operación (-8) – (-10) = 2

	11111000
-	00001010
+	00000010

ÁLGEBRA BOOLEANA

El álgebra Booleana es una herramienta importante porque todas las computadoras digitales requieren una definición de las reglas que rigen la manipulación de variables de dos valores, conceptos dictados por este tipo de álgebra.

Estas reglas se representan por las operaciones básicas AND, OR y NOT. Estas reglas se escriben en forma de tablas de verdad, las cuales especifican los resultados de todas las combinaciones de entrada posibles. Los resultados se expresan en formato de FALSO y VERDADERO. También p uede se como un '1' o un '0'.

La operación Booleana AND se puede expresar con un resultado verdadero sólo si las dos entradas son verdaderas o 1. El símbolo Booleano de la operación AND es un punto (∗), por lo que su representación es

$$C = A * B$$

La tabla de verdad de la operación AND es

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

La operación Booleana OR se puede expresar con un resultado falso sólo si las dos entradas son falsas o 0. El símbolo Booleano de la operación OR es un signo más (+), por lo que se representación es

$$C = A + B$$

La tabla de verdad de la operación OR es

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

La operación Booleana NOT expresa el complemento del valor de entrada. Si la entrada es verdadera la salida es falsa. El símbolo Booleano de la operación NOT es una barra en la parte superior de la variable, por lo que su representación es

$$C = \overline{A}$$

La tabla de verdad de la operación NOT es

A	C
0	1
1	0

Hay otra operación que se usa con frecuencia que es el EXCLUSIVE-OR. Esta operación se puede expresar con un resultado verdadero sólo cuando las dos entradas son diferentes. El símbolo Booleano de la operación EXCLUSIVE-OR es un signo más dentro de un círculo ($\oplus$), por lo que su representación es

$$C = A \oplus B$$

La tabla de verdad de la operación EXCLUSIVE - OR es

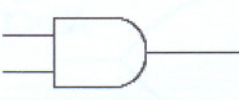
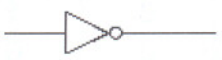
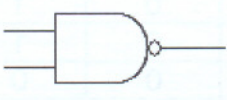
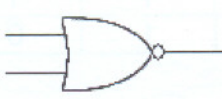
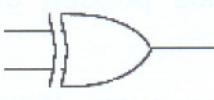
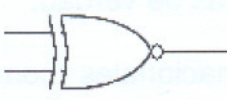
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Las operaciones Booleanas respetan el formato original de la ley conmutativa, asociativa y distributiva.

El álgebra Booleana es un álgebra que se ocupa de variables binarias y operaciones lógicas. Una función Booleana expresa la relación lógica entre

variables binarias y que evalúa todos los valores binarios de la expresión para todos los posibles valores de entrada.

Una función Booleana se puede transformar de una expresión algebraica a un diagrama de circuitos hechos con compuertas lógicas.

Una función Booleana se representa de la siguiente forma:

$$F_1 = A'B'C + A'BC + AB'$$

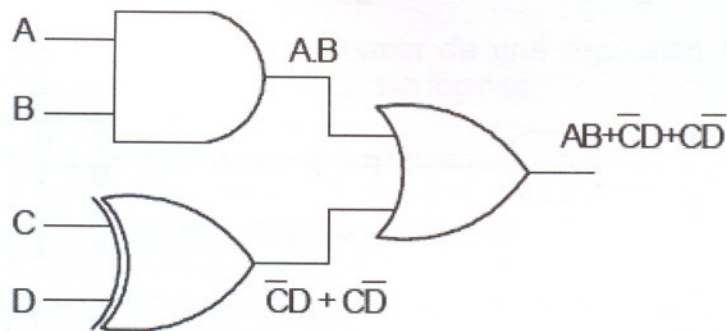
donde el apóstrofe se toma como complemento u operación NOT, el signo más es una operación OR y las variables se operan con una operación AND.

COMPUERTAS LÓGICAS

Una manera generalizada de representar las funciones Booleanas es el uso de símbolos llamados compuertas lógicas digitales. Estas compuertas representan bloques funcionales que reciben entradas, se procesan y generan una salida.

Los símbolos de las compuertas lógicas digitales son los siguientes:

Nombre	Símbolo gráfico	Función algebraica	Tabla de verdad
AND	X, Y → F	F = X Y	X Y | F: 0 0 | 0; 0 1 | 0; 1 0 | 0; 1 1 | 1
OR	X, Y → F	F = X + Y	X Y | F: 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 1
Inversor	X → F	F = X'	X | F: 0 | 1; 1 | 0
Búfer	X → F	F = X	X | F: 0 | 0; 1 | 1
NAND	X, Y → F	F = (X Y)'	X Y | F: 0 0 | 1; 0 1 | 1; 1 0 | 1; 1 1 | 0
NOR	X, Y → F	F = (X + Y)'	X Y | F: 0 0 | 1; 0 1 | 0; 1 0 | 0; 1 1 | 0
XOR (OR exclusivo)	X, Y → F	F = XY' + X'Y F = X⊕Y	X Y | F: 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 0
NXOR	X, Y → F	F = XY + X'Y' F = (X⊕Y)'	X Y | F: 0 0 | 1; 0 1 | 0; 1 0 | 0; 1 1 | 1

Como ejemplo, consideremos la siguiente función:

A	B	C	D	A∗B	C⊕D	SALIDA
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	1	1
1	0	1	0	0	1	1
1	0	1	1	0	0	0
1	1	0	0	1	0	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	0	1

CIRCUITOS COMBINACIONALES Y SECUENCIALES

Los circuitos lógicos para sistemas digitales pueden ser combinacionales y secuenciales.

Un circuito combinacional consiste en compuertas lógicas cuyas salidas en cualquier momento, estarán determinadas por la combinación actual se entradas, quedando definida su operación por un conjunto de funciones Booleanas, por lo que se puede representar en términos de tablas de verdad.

Un circuito secuencial usa elementos combinacionales (compuertas lógicas) más elementos de almacenamiento (memorias), y sus salidas son función de la combinación actual de entradas y del estado de los elementos de almacenamiento, ya sea en el momento actual o como resultado de estados anteriores.

Capítulo 5

Motores

Motores

DEFINICIÓN

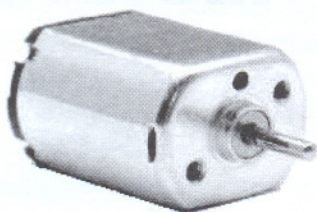
En el Capítulo 1 el concepto Mecatrónica se definió y se trató con amplitud y entre más se avanza en el estudio de este libro, más claro debe quedar el tema del que se trata. Aun así, aunque es una palabra interesante, puede resultar un poco confusa o muy técnica para muchas personas que no están familiarizadas con el tema. Entonces podemos, aunque sea durante este capítulo, buscar un sinónimo a la palabra y en este caso será Robótica.

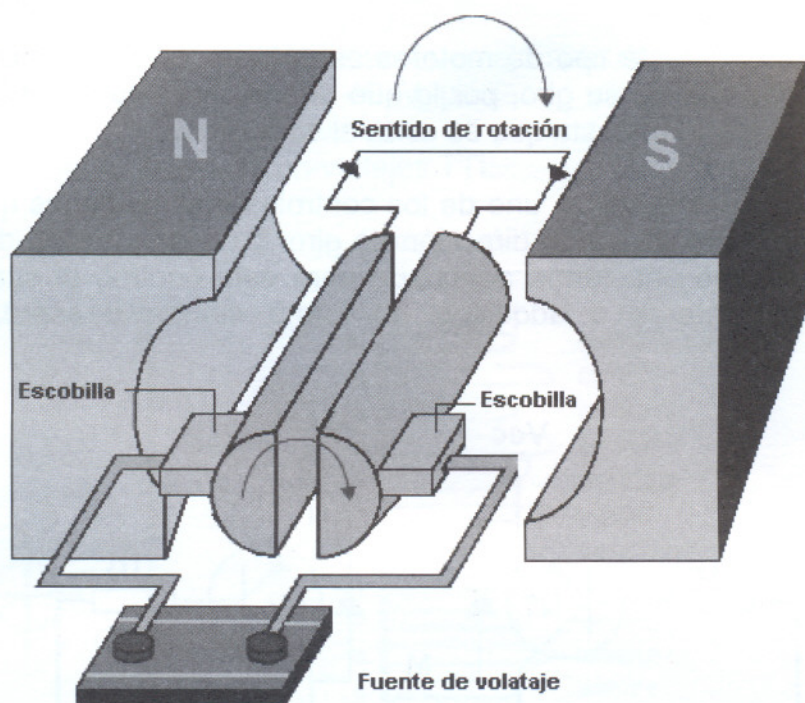
La robótica es una actividad técnico-científica dedicada al desarrollo y construcción de dispositivos cuya finalidad es realizar tareas en la misma forma en la que son realizadas por las personas. Esta imitación de tareas puede ser sencilla o muy compleja, por lo que una de las características más impresionante puede ser el movimiento que realizan los robots. De aquí que se diga que “Si no se mueve, no es un robot”

El movimiento que se pueda lograr en los robots sólo es posible a través de un motor, o varios, conectado a una serie de poleas, engranes o de cualquier otro mecanismo mecánico que ayude al movimiento del elemento del robot.

Por lo tanto, vale la pena revisar los fundamentos de operación de los motores que nos permitan introducirnos en este fascinante tema.

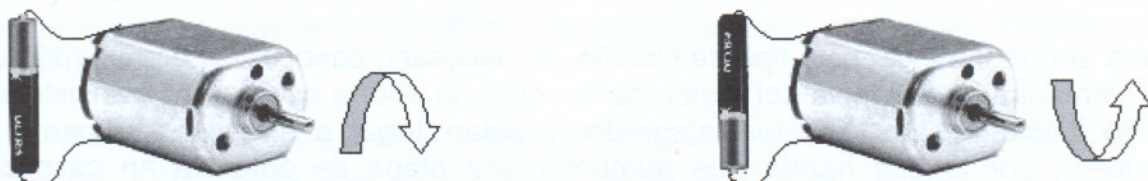
MOTORES DE CORRIENTE DIRECTA (DC)

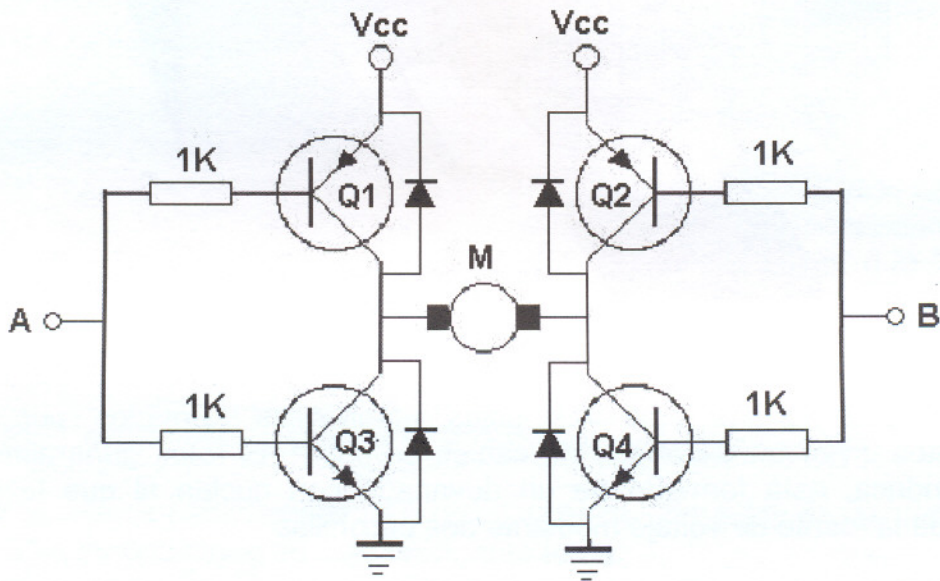
Los motores de DC son dispositivos tecnológicos perfectamente estudiados que combinan eficiencia y una operación controlada con facilidad. Los primeros estudios y pruebas que se realizaron para desarrollarlos se hicieron en 1821. Un motor de corriente continua es un dispositivo mecánico que transforma la energía eléctrica en movimiento rotacional. La principal característica del motor de corriente continua es su capacidad de regular su velocidad desde cero hasta la máxima diseñada.

Pensando en un prototipo que se quisiera desarrollar, la imagen que se muestra podría ser lo más representativo de estos dispositivos.

Los motores de corriente continua están formados por dos partes principales: el estator y el rotor, concepto que se aclara en la siguiente imagen

El estator está formado por los polos magnéticos formados por imanes permanentes o por un devanado de cables de cobre. El rotor, generalmente de forma cilíndrica, está formado de un devanado con núcleo al que le llega la corriente de la fuente de voltaje mediante dos escobillas.

La magnitud de la fuente de voltaje va a determinar la velocidad del motor medida en revoluciones por minuto (RPM).

Los motores de DC podrán varias el sentido del giro dependiendo del sentido del flujo de corriente que se les haga pasar, es decir, como son motores de corriente continua, la fuente de voltaje que los alimenta determinará el sentido, siempre y cuando la fuente de voltaje este en el rango de operación del motor de DC.

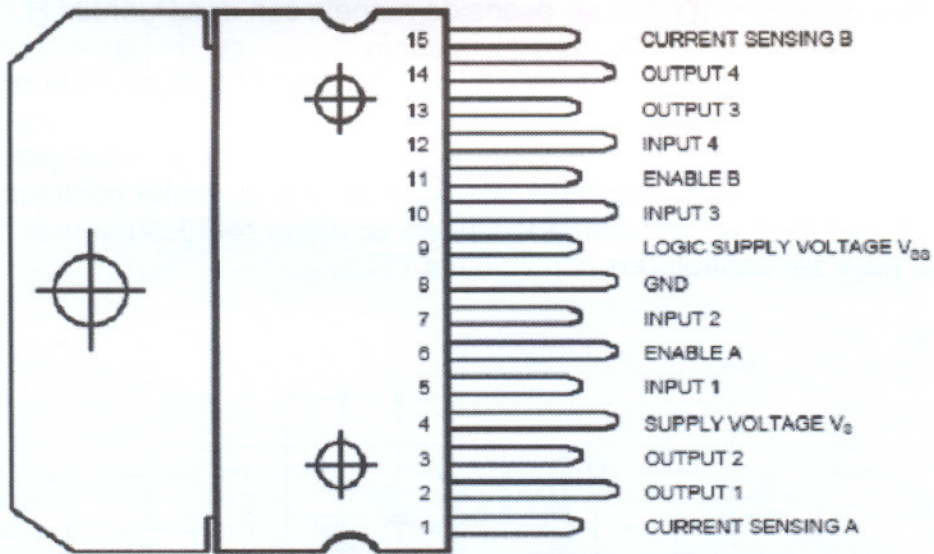
En una descripción más gráfica, el sentido de giro del eje del motor depende de la posición de la batería en los bornes del motor, lo que se puede representar de la siguiente forma:

Una característica de este tipo de motores es que, por si solos, no cuentan con detectores de posición en su giro, por lo que una vez que se les aplica el voltaje giran a máxima velocidad hasta que se corta el voltaje.

Con base a esta característica, uno de los controles más comunes que se puede tener sobre estos motores es la dirección de giro. La configuración de un circuito llamado Puente-H es una forma adecuada para este control, el cual puede ser armado con transistores o adquirido en forma comercial como un circuito integrado.

La lógica de operación de este tipo de circuitos es que los transistores trabajan en las regiones de corte y saturación de los transistores, por lo que el sentido de giro del motor obedece a las reglas de la siguiente tabla:

A	B	Dirección	
0	0	Sin giro	
0	1	Derecho	
1	0	Izquierdo	
1	1	Sin giro	No se recomienda

Para armar con éxito este tipo de circuito, es necesario conocer las características de demanda de potencia del motor, de lo contrario podría quemar los transistores o no funcionar. Los circuitos integrados pueden llegar a manejar fácilmente 1 Ampere, por lo que habría que aumentar una etapa de potencia en caso de necesitar una mayor potencia.

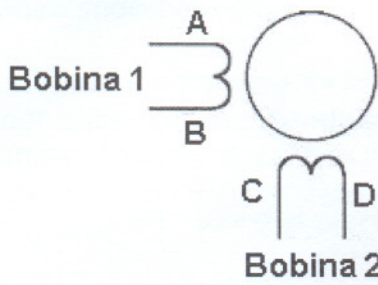
El circuito integrado modelo L298N es un buen ejemplo de un Puente H que sirve para manejar motores de 4.5 V – 46 V pudiendo proporcionar una corriente continua hasta de 4 amperes. El chip tiene 15 patas con dos puentes completos diseñados para ser controlados con voltajes TTL.

La distribución de pines según la hoja de datos del fabricante es

Otra forma de obtener el control sobre la dirección de giro del eje del motor de DC es usar relevadores que simulen la inversión de voltaje en los bornes del motor (El kit K-415 ofrece esta posibilidad).

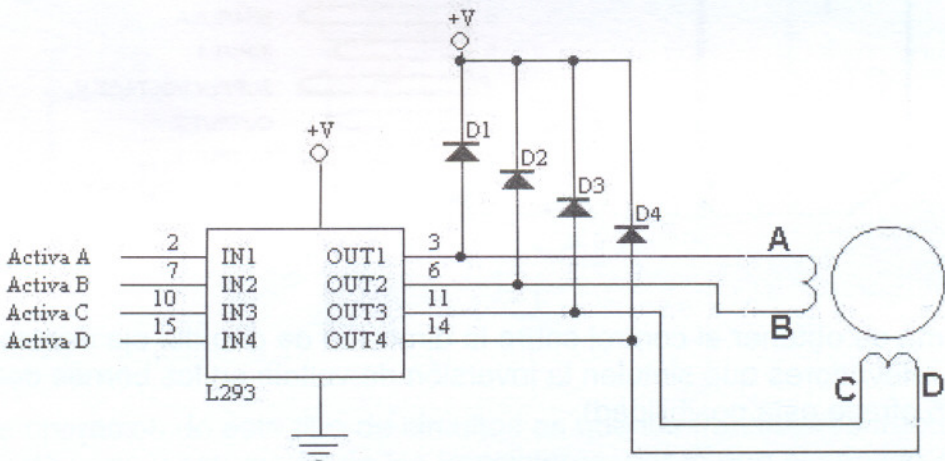
MOTORES DE PASOS

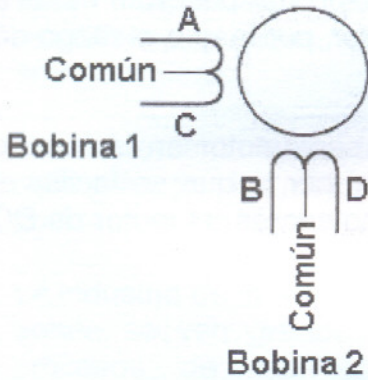
Los motores de pasos son dispositivos mecánicos que ofrecen un gran control y precisión en su movimiento rotacional, pues como su nombre lo indica, se pueden mover en pequeños pasos, en pequeños incrementos que se controlan con pulsos de voltaje.

Estos pasos en su giro pueden variar desde 90° hasta pequeños movimientos de 1.8°. Para lograr giros de 90° se requieren cuatro pasos para dar una vuelta completa. Para lograr giros de 1.8° se requieren doscientos pasos para lograr una vuelta completa.

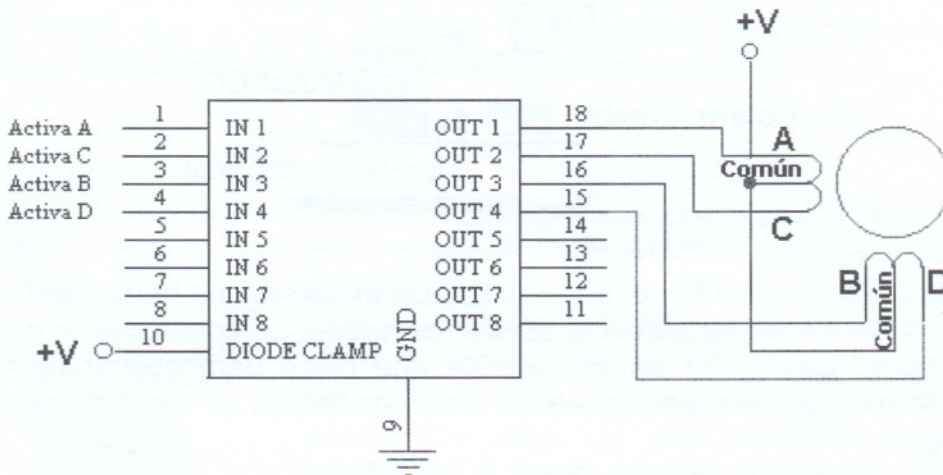
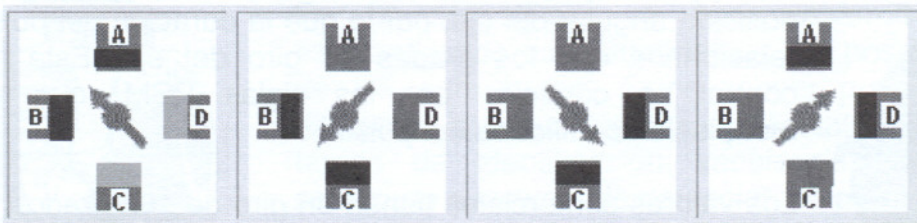
Esto motores están formados por diferentes bobinas que, al ser energizadas en una secuencia determinada, van a generar los giros en el eje del motor.

Uno de los dos tipos de motores de paso es el Bipolar. La forma de identificarlo es que solamente tiene cuatro cables de alimentación.

Cada bobina se puede pensar como un motor de DC, por lo que estos motores de pasos pueden ser manejados con un circuito de Puente H por cada bobina. Así, un motor de pasos de maneja con dos Puentes H.

El circuito integrado L293D es un buen ejemplo de un Puente H que sirve para manejar motores de paso bipolar de baja potencia en un rango de 4.5 V – 36 V pudiendo proporcionar una corriente continua hasta de 600 mA. El chip tiene un encapsulado DIP de 16 patas con dos puentes completos diseñados para ser controlados con voltajes TTL.

La tabla de operación de ente circuito es:

Pasos	Terminales			
	A	B	C	D
1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	1	0

El otro tipo de motores de paso es el Unipolar. La forma de identificarlo es que tiene seis cables de alimentación.

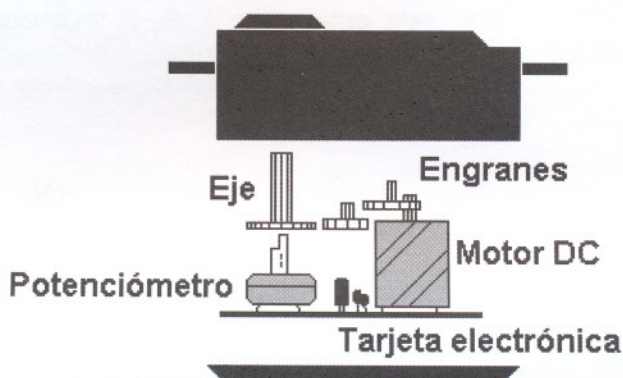
Uno de los circuitos integrados de baja potencia usado para controlar este tipo de motores de pasos puede ser el ULN2803, que es un arreglo de ocho transistores que pueden manejar una corriente máxima de 500 mA.

Al igual que en el circuito anterior, la activación es en niveles TTL y es a través de las entradas de activación A, B, C y D, como se muestra en la siguiente figura

La secuencia se puede representar considerando las posiciones de las bobinas y su activación, lo que causa el movimiento del eje del motor. Las bobinas de color negro representan las energizadas, lo que produce el movimiento en el eje representado por la flecha.

En esta secuencia se muestra una rotación completa, la cual puede lograr en un número diferente de pasos.

SERVOMOTORES

Los servomotores son motores de DC que en el mismo cuerpo del dispositivo cuentan con un sensor de posición el cual permite identificar las posiciones angulares del eje. En muchas versiones, este sensor de posición es una resistencia variable la cual se convierte en el eje del motor, por lo que el rango de giro puede quedar limitado al rango del potenciómetro.

La siguiente figura ilustra las partes principales de un servomotor en la que se aprecia el juego de engranes que reduce el giro del eje, por lo que se facilita el control de posición, ya que como se dijo en secciones anteriores, el motor de DC solamente tiene control de giro cambiando su polaridad.

Engranes
Eje
Motor DC
Potenciómetro
Tarjeta electrónica

Esta última característica hace, de este tipo de motores, una herramienta muy útil en aplicaciones típicas de robótica, donde los rangos de operación se encuentran limitados como podría ser en el caso de una pinza para sujetar objetos. Los valores máximos comerciales se encuentran por debajo de los 220 grados en el rango de giro.

La tarjeta electrónica cuenta con los elementos necesarios para medir y corregir la posición angular del eje. Esta tarjeta cuenta con tres cables, uno de voltaje de alimentación, otro de tierra y uno más de control.

El cable de control recibirá pulsos de duración variable que será el tiempo en que el motor va a estar prendido y así cambiará la posición angular del eje, por lo que la duración del pulso estará relacionada con los grados de giro del eje. Esta forma de control es conocida por las siglas PCM, que significan modulación codificada del pulso.

Los valores comerciales tienen un giro de 180 grados asociado a un pulso de duración de 2.5 milisegundos, de donde se deduce que un pulso de 0.5 milisegundos daría un giro de cero grados y uno de 1.5 milisegundos daría un giro de 90 grados.

ROBOTICA

Robótica es un término que se explica por si sólo y que se aplica a esa inquietud humana por desarrollar dispositivos que simulen tener vida e inteligencia artificial. Estas características son usadas para desarrollar un sinnúmero de actividades repetitivas, pesadas, de alto riesgo para las personas o simplemente para entretenimiento.

Invariablemente, estas tareas se realizan a través del control de los movimientos realizados por los motores del robot o por cualquier otro tipo de actuadores, como podrían ser los hidráulicos

La industria de la manufactura ha sido un buen campo de desarrollo de la robótica donde se ven grandes beneficios en el control de la automatización de los procesos de fabricación de productos aumentando, principalmente, la productividad de las industrias.

Aunque la creación de autómatas ha sido una actividad humana desde hace muchos años, fue a partir de 1980 que empezó el crecimiento de estas tecnologías y ha venido teniendo un crecimiento exponencial.

En la actualidad, la robótica ocupa un lugar importante en muchas actividades productivas y de entretenimiento. En la automatización industrial, se usa en tareas repetitivas logrando grandes volúmenes de producción, en la programación selectiva de actividades para lograr volúmenes pequeños de producción pero de una variedad importante de productos y para generar diferentes cadenas de producción con la finalidad de lograr productos complejos en poco tiempo, comparado contra una mano de obra calificada.

En la agricultura, se han desarrollado robots especializados para automatizar las pesadas labores de los agricultores. Estos robots pueden detectar diferentes plantas, hierbas o vegetales y desarrollar tareas de identificación, recolección, clasificación, sembrado e incluso, de detección de plagas.

En la actividad aero-espacial hay grandes avances, tanto que ya hay varios aeropuertos en el mundo que están certificados para el control automático de despegue y aterrizaje de aviones y en la carrera espacial los robots teledirigidos están desarrollando tareas de alto riesgo y complejidad para los astronautas.

En el área de seguridad y combate al vandalismo, la policía va teniendo dispositivos de ayuda que les permite obtener información de zonas internas de alto riesgo mediante el envío de información de video, audio y otras a los policías que se encuentran en el exterior.

Y quedan muchas más aplicaciones por platicar y crear, por lo que es el momento de seguir avanzando en el estudio de la mecatrónica y usar esta disciplina para la innovación de nuevos sistemas de automatización y control.

Capítulo 6

Microprocesadores

Microprocesadores

DEFINICIÓN

Los microprocesadores son circuitos integrados que desde el año 1969 con la compañía Intel, revolucionaron el diseño electrónico de dispositivos. El 4004 fue el primer microprocesador que tenía una unidad central de procesos (CPU) de 4 bits, una memoria ROM para almacenar instrucciones, una memoria RAM para almacenar programas y aplicaciones de los usuarios y algunos puertos de entrada y salida. Este microprocesador fue diseñado para calculadoras de bolsillo.

El cambio en el diseño electrónico radica principalmente en que los diseños anteriores a los microprocesadores eran específicos para las funciones que eran diseñados, mientras que los diseños con microprocesadores permiten utilizar el mismo hardware para diferentes funciones, funciones que quedan desarrolladas y controladas por la programación del microprocesador.

En la actualidad contamos con un sinnúmero de dispositivos electrónicos que simplifican tareas, ofrecen diferentes servicios y nos proporcionan diversión.

Este capítulo describe, principalmente, el uso de los microprocesadores en las computadoras personales describiendo su arquitectura, sus principales comandos, su uso dentro de las computadoras personales y las aplicaciones que se pueden desarrollar como controlador de proyectos mecatrónicos.

La computadora se puede representar en algunos bloques funcionales para entender su forma de operación. Por el momento, son tres los principales bloques: Primero está el microprocesador que está formado por la Unidad Aritmética Lógica (ALU) y el módulo de control, representado por los *buses* de datos, direcciones y control y por la electrónica que ejerce control entre las diferentes unidades funcionales. El segundo bloque está representado por la memoria RAM, memoria en la que, según el modelo de computadora de John von Neumann, almacena los datos y los programas de aplicaciones del usuario. El tercer bloque está formado por los puertos de entrada y salida, puertos que permiten la comunicación del microprocesador con el resto de la electrónica, tanto de la computadora (teclado, *mouse*, pantalla, disco duro, etc.) como de los circuitos externos.

El siguiente diagrama muestra los principales bloques funcionales de un modelo general de computadora.

E/S

Entrada

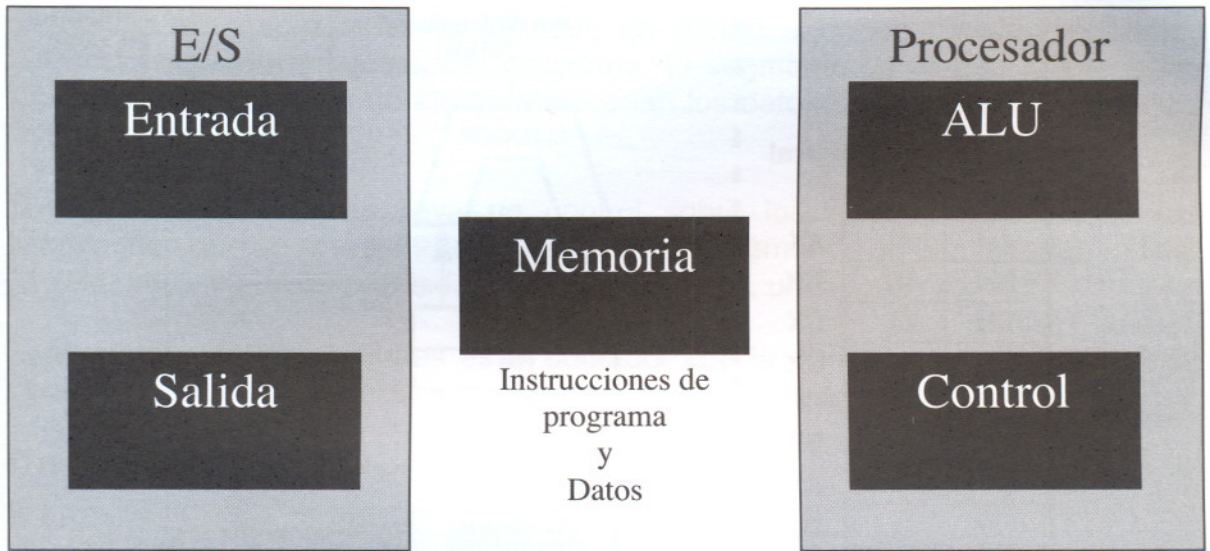
Salida

Memoria

Instrucciones de programa y Datos

Procesador

ALU

Control

ARQUITECTURA DE MICROPROCESADORES

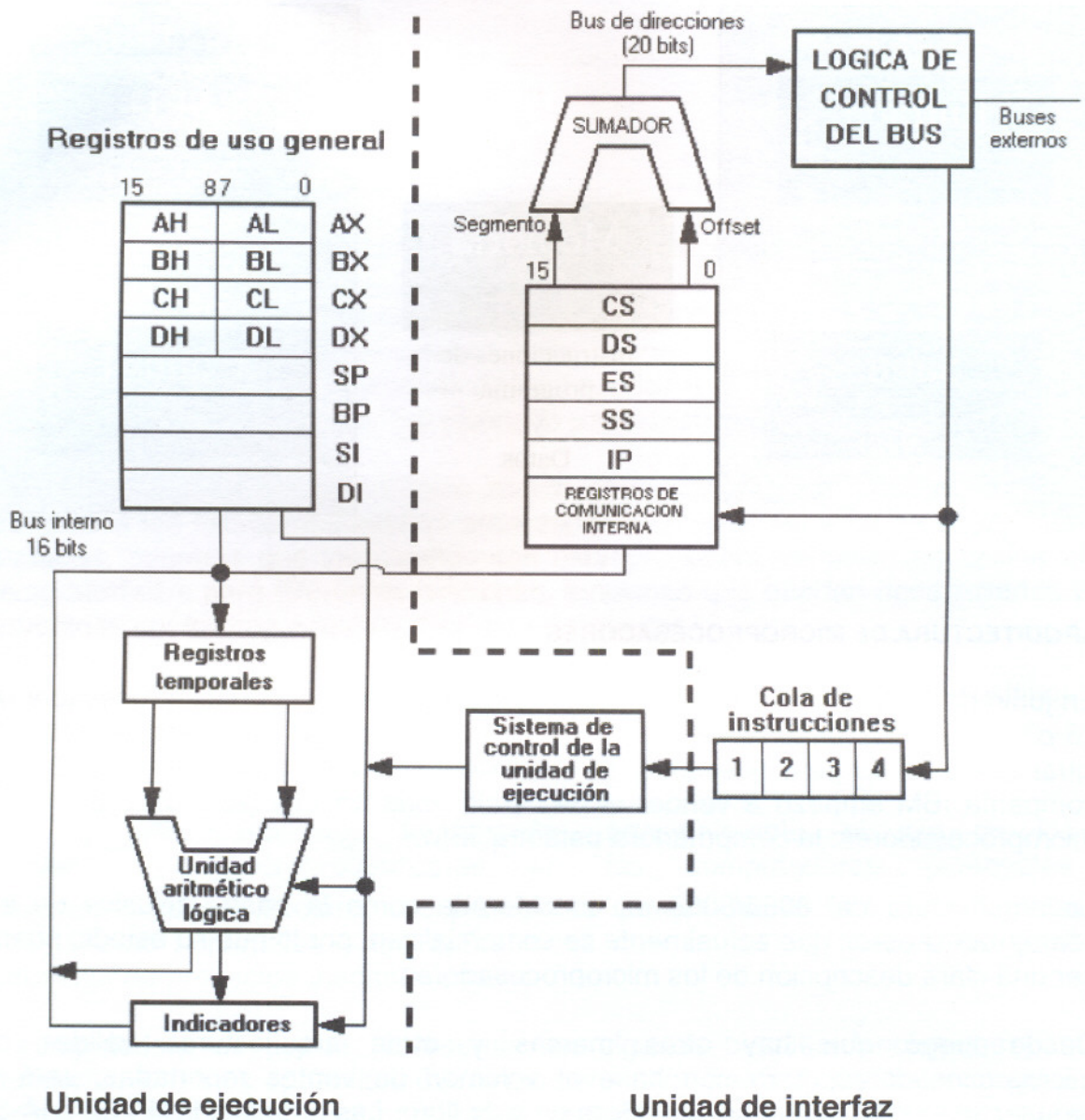
En junio de 1978 la compañía Intel lanzó al mercado el primer microprocesador de 16 bits, al que llamó 8086. Un año más tarde lanzó el 8088, que internamente era igual que el 8086 pero que tenía un bus externo de datos de 8 bits. En 1981 la compañía IBM empezó a vender el desarrollo más notable para esta familia de microprocesadores: la computadora personal IBM.

La arquitectura del 8086/8088 es considerada como la base, inclusive de los microprocesadores que actualmente se comercializan, por lo que su estudio puede ser una clara descripción de los microprocesadores.

Desde luego que hay otras marcas y otras arquitecturas válidas de microprocesadores, pero con base al volumen de ventas reportadas, será la arquitectura IA-16 la que se considere en este libro. Las arquitectura IA-32 e IA-64 operan bajo los mismos conceptos, sólo que el tamaño de la palabra es diferente al igual que el número de instrucciones disponibles, instrucciones enfocadas a multimedia, comunicaciones y seguridad.

El 8086 es un microprocesador de 16 bits, pudiendo dividir su estructura en dos bloques: la unidad de ejecución y la unidad de interfaz. La unidad de ejecución es la encargada de realizar todas las operaciones, mientras que la unidad de interfaz del bus es la encargada de comunicación de datos e instrucciones al mundo exterior.

Esta división tiene ventajas y ahorros en el diseño de una interfaz de 32, 16 u 8 bits. Se muestra una representación en la siguiente figura.

Unidad de ejecución **Unidad de interfaz**

Para fines prácticos, el modelo de microprocesador que nos interesa es aún más simple, ya que sólo consideraremos los registros lógicos que se van a programar. Este modelo se basa en cuatro grupos: registros de propósito general, de segmento de datos, de banderas y el apuntador de instrucciones.

En los registros de propósito general se manipulan los datos, los bytes, según los comandos del programa y se pueden comparar a las variables que se usan en los lenguajes de alto nivel.

Los registros de segmentos marcan la dirección en memoria del segmento de datos, el de código y el de la pila. El segmento de datos se refiere a la localidad en memoria donde se almacenan los datos que manipula el programa. El segmento

de código se refiere a las localidades en memoria donde se almacenan las instrucciones, los comandos, del programa. El segmento de pila se refiere a las localidades en memoria donde se almacenan los datos de la pila a través de las instrucciones PUSH y POP.

El registro de banderas lleva un control sobre los resultados de todas las operaciones que se realizan. Estos resultados, en términos de bits, pueden indicar un valor cero, un bit de paridad, un cambio de signo, un bit acarreo, etc.

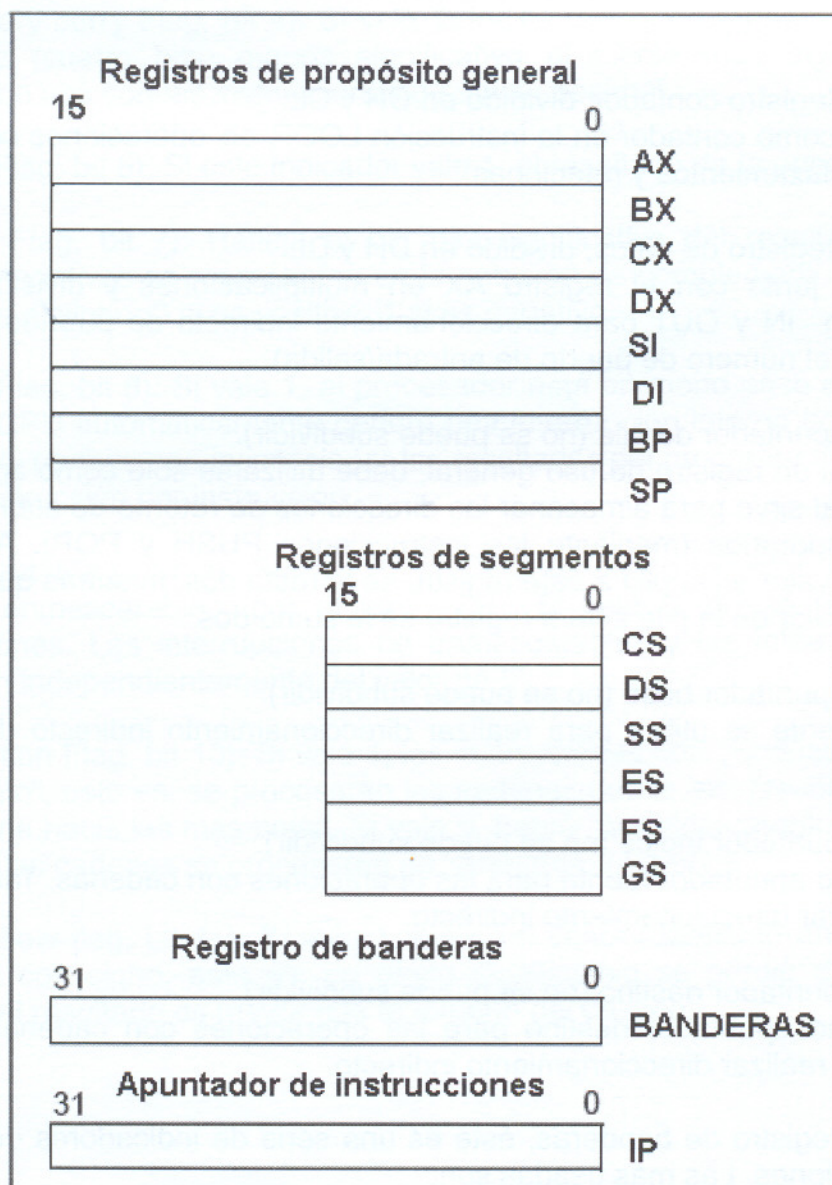
El apuntador de instrucciones es el contador con la dirección en memoria de la siguiente instrucción a ejecutar, registro que lleva la unidad de control.

El modelo lógico del microprocesador es el siguiente:

Registros de propósito general
15 0
AX
BX
CX
DX
SI
DI
BP
SP

Registros de segmentos
15 0
CS
DS
SS
ES
FS
GS

Registro de banderas
31 0
BANDERAS

Apuntador de instrucciones
31 0
IP

Los registros de propósito general se pueden utilizarse como fuente o destino en operaciones aritméticas y lógicas y también pueden tener algunas funciones determinadas por el conjunto de instrucciones del microprocesador. A continuación se ofrece una breve explicación:

- AX = Registro acumulador, dividido en AH y AL (8 b its cada uno).
Es un registro sugerido para almacenar el resultado de las operaciones realizadas. Hay instrucciones como IN y OUT, que son instrucciones de comunicación a puertos, que trabajan con AX o con uno de sus dos bytes (AH o AL). También se utiliza este registro (junto con DX a veces) en multiplicaciones y divisiones.

- BX = Registro base, dividido en BH y BL.
Es el registro base de propósito para direccionamiento de memoria en sus diferentes formatos

- CX = Registro contador, dividido en CH y CL.
Se utiliza como contador en la instrucción LOOP, en operaciones con cadenas y en desplazamientos y rotaciones

- DX = Registro de datos, dividido en DH y DL.
Se utiliza junto con el registro AX en multiplicaciones y divisiones, en la instrucción IN y OUT para direccionamiento indirecto de puertos (el registro DX indica el número de puerto de entrada/salida).

- SP = Apuntador de pila (no se puede subdividir).
Aunque es un registro de uso general, debe utilizarse sólo como apuntador de pila, la cual sirve para almacenar las direcciones de retorno de subrutinas y los datos temporarios (mediante las instrucciones PUSH y POP). Al introducir (push) un valor en la pila a este registro se le resta dos, mientras que al extraer (pop) un valor de la pila este a registro se le suma dos.

- BP = Apuntador base (no se puede subdividir).
Generalmente se utiliza para realizar direccionamiento indirecto dentro de la pila.

- SI = Apuntador índice (no se puede subdividir).
Sirve como apuntador fuente para las operaciones con cadenas. También sirve para realizar direccionamiento indirecto.

- DI = Apuntador destino (no se puede subdividir).
Sirve como apuntador destino para las operaciones con cadenas. También sirve para realizar direccionamiento indirecto.

Respecto al registro de banderas, éste es una serie de indicadores del resultado de las operaciones. Las más usadas son:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Flag	--	--	--	--	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

CF (Carry Flag, bit 0): Si vale 1, indica que hubo "acarreo" (en caso de suma) hacia, o "préstamo" (en caso de resta) desde el bit de orden más significativo del resultado. Esta bandera es usada por instrucciones que suman o restan números que ocupan varios bytes. Las instrucciones de rotación pueden aislar un bit de la memoria o de un registro poniéndolo en el CF.

PF (Parity Flag, bit 2): Si vale uno, el resultado tiene paridad par, es decir, un número par de bits a 1. Esta bandera se puede utilizar para detectar errores en transmisiones.

AF (Auxiliary carry Flag, bit 4): Si vale 1, indica que hubo "acarreo" o "préstamo" del nibble (cuatro bits) menos significativo al nibble más significativo. Este indicador se usa con las instrucciones de ajuste decimal.

ZF (Zero Flag, bit 6): Si este indicador vale 1, el resultado de la operación es cero.

SF (Sign Flag, bit 7): Refleja el bit más significativo del resultado. Como los números negativos se representan en la notación de complemento a dos, este bit representa el signo: 0 si es positivo, 1 si es negativo.

TF (Trap Flag, bit 8): Si vale 1, el procesador está en modo paso a paso. En este modo, la CPU automáticamente genera una interrupción interna después de cada instrucción, permitiendo inspeccionar los resultados del programa a medida que se ejecuta instrucción por instrucción.

IF (Interrupt Flag, bit 9): Si vale 1, la CPU reconoce pedidos de interrupción externas enmascarables (por el pin INTR). Si vale 0, no se reconocen tales interrupciones. Las interrupciones no enmascarables y las internas siempre se reconocen independientemente del valor de IF.

DF (Direction Flag, bit 10): Si vale 1, las instrucciones con cadenas sufrirán "auto-decremento", esto es, se procesarán las cadenas desde las direcciones más altas de memoria hacia las más bajas. Si vale 0, habrá "auto-incremento", lo que quiere decir que las cadenas se procesarán de "izquierda a derecha".

OF (Overflow flag, bit 11): Si vale 1, hubo un desbordamiento en una operación aritmética con signo, esto es, un dígito significativo se perdió debido a que el tamaño del resultado es mayor que el tamaño del destino.

MEMORIA RAM

Este modelo lógico del microprocesador interactúa con la memoria RAM (*random access memory* = memoria de acceso aleatorio), la cual se puede entender como la memoria de trabajo en la que se encuentran todos los datos, tanto el código del programa como los datos de la aplicación en cuestión.

La memoria RAM puede ser del tipo estática o dinámica, lo que es definido por el diseño técnico. En lo que se refiere al uso entre una y otra es transparente para el usuario, aunque las memorias dinámicas son más baratas, consumen menos energía, pueden ser de tamaño físico menor y contener una mayor cantidad de memoria. Sus desventajas, en comparación a las estáticas, son más lentas y requieren de circuitos adicionales para generar la señal de refresco. Las dos clases de memoria son volátiles, lo que significa que su contenido se pierde si se apaga la fuente de energía.

Los sistemas modernos contiene poca memoria RAM estático que se emplea en las partes de acceso de gran velocidad como es el caso de la memoria caché.

La memoria se puede modelar como una pila de registros de memoria de 8 bits, es decir, de un byte. Esta pila de bytes se numera del 0 hasta el final de la capacidad de la memoria. En el caso del microprocesador 8086, el bus de direcciones es de 16 bits, por lo que la capacidad máxima del bloque de direccionamiento de memoria es de 64 KByte (65,536 bytes).

Registros de direcciones de memoria (MAR)

Byte n	7	6	5	4	3	2	1	0
	7	6	5	4	3	2	1	0
	..	..	..	..	..	..	..	..
	..	..	..	..	..	..	..	..
	7	6	5	4	3	2	1	0
Byte 0	7	6	5	4	3	2	1	0

Celdas individuales de memoria

Registros de datos de memoria (MDR)

El registro de direcciones de memoria contiene la dirección que debe accesarse para obtener cada uno de los datos. El registro de direcciones de memoria está conectado a un decodificador que interpreta la dirección y la activa para su uso, tanto de lectura como de escritura. Los grupos de celdas de bits, los bytes, contienen una línea independiente, lo que determina los n elementos de la memoria, estableciéndose el número total con la ecuación 2^n líneas.

El registro de datos de memoria está diseñado de modo que se conecta adecuadamente a todas las celdas en la unidad de memoria. Todos sus bits están conectados al circuito de lectura que garantiza que sólo un grupo de celdas se activa en un momento determinado.

Debido a que el 8086 es un microprocesador con un bus de datos y de direcciones de 16 bits, el máximo bloque de direcciones de memoria que puede leer simultáneamente es de 64KBytes (65,536 bytes). Para que pueda leer un bloque de memoria de 1 MByte (1,048,576 bytes), se necesitan definir 16 bloques de 64KBytes.

Para implementar esta idea, las direcciones físicas de memoria se representan en un formato lógico y así poder trabajar con ellas. De esta manera, la memoria se representa con dos bloques: el primero con una dirección que indica el origen del segmento más una segunda dirección que sería la serie de direcciones que indican el desplazamiento sobre el origen.

El formato general sería de la forma 1000:0000, donde los 4 dígitos de la izquierda marcan el origen de los 16 bloques diferentes y los 4 dígitos de la derecha marcan el desplazamiento, siendo expresado siempre en números hexadecimales.

Por ejemplo, la dirección lógica 1234:4321 tiene una dirección física en la pila de bytes de memoria de 16661H, lo cual se obtiene al rotar a la izquierda cuatro bits el número segmento (equivalente a un dígito) y luego sumándole el valor del desplazamiento.

```
   1234:4321

   12340
+   4321
---------
   16661
```

Esto implica que hay muchas formas de expresar en formato lógico la misma dirección física. La dirección 1234:4321 también podría ser 1666:0001 ó 1665:0011 ó 1664:0021, etc.

Una segunda consideración que hay que tomar en cuenta es la forma y posición en la que se almacenan los datos. Como la memoria es una pila de bytes y algunos números pueden ser de 16 o 32 bits, el dato completo siempre va a estar en direcciones adyacentes, solamente que es en orden inverso.

Para una palabra (16 bits) la escritura es de la siguiente forma:

Orden ascendente ⊠

9C	E6

Palabra E69CH

Para una palabra doble (32 bits) la escritura es de la siguiente forma:

Orden ascendente ⊠

4A	5B	00	12

Palabra doble 12005B4AH

PUERTOS

Independientemente de las capacidades y potencia del microprocesador, de su facilidad para trabajar con la memoria RAM y del tamaño de la misma memoria, la verdadera utilidad de una computadora radica en sus capacidades de entrada y salida. Estas entradas y salidas son la forma de comunicación que se tiene con el mundo exterior y sin ellas no habría dispositivos periféricos a través de los cuales hacemos uso de las computadoras.

El teclado, el *mouse*, la pantalla, la impresora, el escáner, las memorias USB, el disco duro, etc., son ejemplos de dispositivos periféricos, los cuales se clasifican en dispositivos de entrada y salida, según el flujo de información a través de ellos.

Los requisitos de cada dispositivo de entrada/salida, considerando la necesidad de ofrecer dispositivos con capacidad de direccionamiento, de sincronización, de estado y de control externo, indican las normas establecidas para poder acoplarlos a las computadoras a través de módulos o circuitos electrónicos con funciones definidas y estos a su vez se comunicarán con los microprocesadores a través de direcciones o puertos perfectamente definidos.

Los puertos son las 'puertas' por donde los datos van a entrar a los registros del microprocesador para ser procesados y posteriormente salir para ser devueltos al dispositivo periférico correspondiente.

En la práctica, es claro que muchos dispositivos estarán conectados al microprocesador, por lo que se debe tener la capacidad de reconocerlos individualmente. Para esto, las computadoras actuales tienen 65,536 puertos diferentes para organizar el flujo de información entre ellos.

Al igual que con las memoria, el acceso a los periféricos, se realiza a través de registros de datos y direcciones, que funcionan en forma similar al MAR y MDR comentados en el acceso a memoria RAM. Así, el microprocesador tiene que indicar si la dirección solicitada es a una dirección de memoria o a una dirección de puerto de E/S.

Capítulo 7

Lenguaje ensamblador

Lenguaje ensamblador

DEFINICIÓN

Los lenguajes de programación son la herramienta con que se cuenta para indicar a las computadoras la secuencia de instrucciones o comandos para realizar las diferentes funciones que desarrollan.

Estos lenguajes tienen diferentes características según el nivel en el que interactúan con la computadora. El nivel más bajo de programación es tratar directamente con el microprocesador, el cual trabaja en código binario. El lenguaje ensamblador es la codificación que permite trabajar a este nivel de programación. Las instrucciones escritas en código binario se conocen con el nombre de código de máquina. Escribir programas en este código es un poco tedioso y requiere de habilidades especiales. El lenguaje ensamblador es una codificación especial que permite agrupar en comandos cortos y con cierto sentido este código de máquina. Esta codificación es la que forma el lenguaje ensamblador y a cada comando también se le llama mnemónico.

Escribir un programa utilizando mnemónicos es más sencillo que el código de máquina porque son una versión abreviada de la operación que realiza una instrucción. También, dado que las instrucciones describen las operaciones del programa, se facilita su comprensión y se reduce la posibilidad de cometer errores. Sin embargo, esta lista de comandos todavía tiene que convertirse en código de máquina ya que es el formato reconocido por el microprocesador. Para esta conversión hay dos programas que se usan con mucha frecuencia y que se conocen como compiladores para lenguaje ensamblador.

El primer programa es de grandes capacidades y que ofrece una gran variedad de opciones llamado MASM (*Macro Assembler*, de Microsoft) y el segundo es utilizar un *debugger* incluido en todas las computadoras con sistema operativo Microsoft (DOS o Windows), por lo que lo hace que esté al alcance de todos los usuarios que tengan computadoras con estas características.

El programa se llama DEBUG y es un compilador basado en una arquitectura IA-16 como la del microprocesador 8086. Algunas características importantes de este programa es que puede crear archivos con extensión .COM, lo que permite crear archivos ejecutables por el sistema operativo y por lo mismo crear archivo de un tamaño máximo de 64KBytes. También tiene la característica de que inicia el segmento de codificación del programa en la localidad 100H de desplazamiento a partir del segmento de código indicado en el registro CS.

Debido a las características anteriores, el programa Debug será la herramienta que se usará en la codificación del lenguaje ensamblador.

INTRODUCCIÓN AL PROGRAMA DEBUG

Debug es una forma fácil de empezar a tener contacto con el *hardware* de la computadora. Este programa muestra el contenido de los registros del microprocesador, los registros de la memoria RAM, las instrucciones escritas en los archivos con programas ejecutables y también programar pequeñas rutinas en ensamblador para trabajar directamente con el microprocesador o a través de las funciones del BIOS (*Basic Input Output System*).

Generalmente, el programa Debug se encuentre en la carpeta C:\Windows\Command, y podemos agrupar sus funciones de la siguiente forma:

- Ensamblar pequeños programas
- Ver el código fuente de pequeños programas con su código de máquina
- Ver el registro de banderas del microprocesador
- Ejecutar paso a paso las instrucciones de un programa para ver como cambian los valores de los registros y memoria
- Modificar o insertar nuevos valores en la memoria RAM
- Buscar valores binarios o ASCII en la memoria RAM
- Mover bloques de memoria de una dirección a otra
- Llenar bloques de memoria con ciertos valores
- Leer o escribir sectores de los discos del sistema

Para iniciar una sesión del programa, hay que abrir una ventana DOS, en ocasiones definida como 'Símbolo del sistema' y después escribir el comando *debug*.

Los comandos del Debug se pueden desplegar en la pantalla con el comando de ayuda '?' y se pueden clasificar en cuatro categorías

1. Creación y lectura de programas

A	Ensamblar un programa usando mnemónicos
G	Ejecutar un programa escrito en memoria
R	Mostrar el contenido de los registros del microprocesador
P	Avanzar un procedimiento o *loop*

T Avanzar una instrucción a la vez
U Desensamblar en mnemónicos un programa escrito en memoria

2. Manipulación de memoria
   C Compara dos rangos de memoria
   D Mostrar el contenido de la memoria
   E Editar registros de memoria
   F Llenar un rango de memoria con valores definidos
   M Mover un rango de memoria de una dirección a otra
   S Buscar un valor definido en un rango de memoria

3. Misceláneas
   H Realizar un suma o resta hexadecimal
   Q Salir del programa Debug y regresar al DOS

4. Entrada – Salida
   I Leer un byte desde un puerto
   O Escribir un byte a un puerto
   L Cargar en memoria datos del disco
   W Escribir datos al disco
   N Nombrar un archivo para los comandos L y W

Al iniciar la sesión del programa, los siguientes valores quedan definidos

1. Todos los segmentos de datos se fijan en la parte inferior de la memoria disponible, justo arriba del programa debug.exe
2. El apuntador de instrucciones (IP) se establece en 0100H
3. Debug reserva 256 bytes para la Pila en la parte superior del segmento de memoria
4. Se reservan 64KBytes de memoria
5. En caso de iniciar el Debug cargando un programa en memoria, los registros BX:CX contienen el tamaño del archivo
6. El registro de banderas se define de la siguiente forma:
   NV bandera overflow = 0
   UP bandera de dirección = up
   EI Interrupción habilitada
   PL bandera de signo = positivo
   NZ bandera cero = 0
   NA bandera auxiliar de acarreo = 0
   PO paridad non
   NC bandera de acarreo = 0

ESTRUCTURA DEL LENGUAJE ENSAMBLADOR

En el lenguaje ensamblador las líneas de código, en su formato más simple, constan de dos partes, la primera es el nombre de la instrucción llamado *Operation Code* y la segunda son los parámetros del comando, los cuales se presentan en el formato de primero el destino y después el origen:

```
comando        destino, origen
ADD            AX, 1234
```

Aquí ADD es el comando que se va a ejecutar, suma, AX es el registro destino donde va a quedar el resultado y también el primer valor a operar y 1234 es el segundo valor que se va a operar.

El nombre de las instrucciones, mnemónicos o códigos de operación, están formados por dos, tres o cuatro letras. Algunos comandos no requieren parámetros y otros solamente requieren uno.

Para poder indicar que el valor a operar o que se va a asignar a un registro proviene de una localidad de memoria, se usarán los corchetes para su notación

```
ADD            AX, [1234]
```

Para crear el primer programa y empezar a ilustrar el proceso, a continuación se presenta un programa que suma dos valores.

El primer paso es iniciar el programa Debug.

Para ensamblar un programa se utiliza el comando "A". En forma predeterminada se asignará la localidad 100H como de inicio del programa.

```
- A ⊠
```

Al hacer esto aparecerá en la pantalla la dirección en memoria en formato de CS:0100, que podría ser algo como

```
0CFC:0100 _
```

Donde los cuatro dígitos hexadecimales del lado izquierdo pueden variar y los cuatro dígitos del lado derecho deben ser 0100 hexadecimal. A continuación se escriben las siguientes instrucciones

```
0CF0:0100 mov ax,5        Almacena el valor 5 en AX
0CF0:0103 mov bx,2        Almacena el valor 2 en BX
0FC0:0106 add ax,bx       Suma el valor de BX a AX
0FC0:0108 hlt             Detiene la ejecución
```

Para regresar al *prompt* del Debug se oprime la tecla Enter o Return representada por el símbolo ←

Para ejecutar el programa se utiliza el comando G indicando el origen y el fin de las localidades de memoria usados, lo que permitirá que el programa termine mostrando el valor de los registros y podamos comprobar el resultado de la función programada.

- G=100 108 ←

```
AX=0007  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0CFC  ES=0CFC  SS=0CFC  CS=0CFC  IP=0108   NV UP EI PL NZ NA PO NC
0CFC:0108 F4            HLT
-
```

PROGRAMACIÓN

Programar en lenguaje ensamblador no es muy común, ya que puede resultar muy complejo en rutinas muy largas, Aún así, tiene ventajas que conviene tener presente y usarlas adecuadamente.

- Algunas veces las rutinas escritas en ensamblador pueden ser más rápidas en comparación a las creadas por el compilador.

- Algunas veces las rutinas en ensamblador pueden tener un tamaño menor en comparación con las creadas por el compilador, ya que pueden hacerse para cumplir un algoritmo específico que no tenga necesidad de validaciones redundantes.

- El ensamblador permite programar directamente características particulares del hardware del sistema o de dispositivos externos y que pueden ser difíciles o imposibles de usar desde un lenguaje de alto nivel.

- Aprender a programar en ensamblador sirve para tener una mejor idea de cómo trabajan las computadoras y los dispositivos electrónicos programables.

Cada tipo de CPU entiende su propio lenguaje de máquina. Las instrucciones en lenguaje de máquina son números almacenados como bytes en memoria. Cada instrucción tiene su propio y único código llamado código de operación, mnemónico u *OpCode*. Estos comandos en los microprocesadores Intel de la familia x86 varían en tamaño, pudiendo tener una cantidad entre 0 y 3 parámetros y uniéndose en diferentes grupos funcionales.

INSTRUCCIONES BÁSICAS

El modelo lógico del microprocesador es un grupo de registros que sirven como variables para manipular los datos que tiene que procesar. Por lo tanto, una de las instrucciones básicas y más usadas es la que sirve para asignar valores a los registros, MOV, estando formada por dos operandos: MOV DESTINO, ORIGEN

TRANSFERENCIA DE DATOS

MOV	Mover valores a registros
XCHG	Intercambiar valores entre registros

OPERACIONES ARITMÉTICAS

ADD	Suma
SUB	Resta
MUL	Multiplicación sin signo
DIV	División sin signo
IMUL	Multiplicación con signo de enteros
IDIV	División con signo de enteros
INC	Incremento en 1
DEC	Decremento en 1

OPERACIONES LÓGICAS

AND	Operador 'Y'
OR	Operador 'O'
NOT	Negación o complemento
XOR	Operador 'O exclusivo'

OPERACIONES DE CORRIMIENTO

ROR	Rotar a la derecha
ROL	Rotar a la izquierda
SHR	Recorrer a la derecha
SHL	Recorrer a la izquierda

OPERACIONES DE COMPARACIÓN

CMP	Comparación de dos valores
TEST	Comprobación de 1 bit
CMPS	Comparación de cadenas

OPERACIONES CON LA PILA

POP	Saca de la pila
POPA	Saca todos los registros de la pila
PUSH	Introduce valores a la pila
PUSHA	Introduce todos los registros a la pila

OPERACIONES DE TRANSFERENCIA INCONDICIONAL

JMP	Salto
CALL	Llamada a subrutina
RET	Regresa, fin de subrutina

OPERACIONES DE TRANFERENCIA CONDICIONAL

JZ / JNZ	Salto si la bandera cero
JS / JNS	Salto si la bandera signo
JE / JNE	Salto de la bandera igual
JG / JNG	Salto si la bandera mayor
JLE / JNLE	Salto si bandera menor o igual

OPERACIONES DE ENTRADA / SALIDA

IN	Lectura de puertos
OUT	Escritura de puertos

CONTROL DE CICLOS

LOOP	Ciclo
LOOPNZ	Ciclo si bandera cero

CONTROL DEL PROGRAMA

HLT	Parar la ejecución
NOP	No realizar operación

EJEMPLO DE PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR

Antes de elaborar algún programa complejo, es conveniente analizar y diseñar una solución del problema que se quiere tratar. Para la codificación de programas en lenguaje ensamblador, la metodología de diagramas de flujo es adecuada para realizar el análisis y diseño.

Como primer ejercicio, sumaremos dos números de acceso inmediato y el resultado se almacenará en la localidad de memoria 120H del segmento actual de datos.

Inicio

Carga el valor de memoria en el registro base

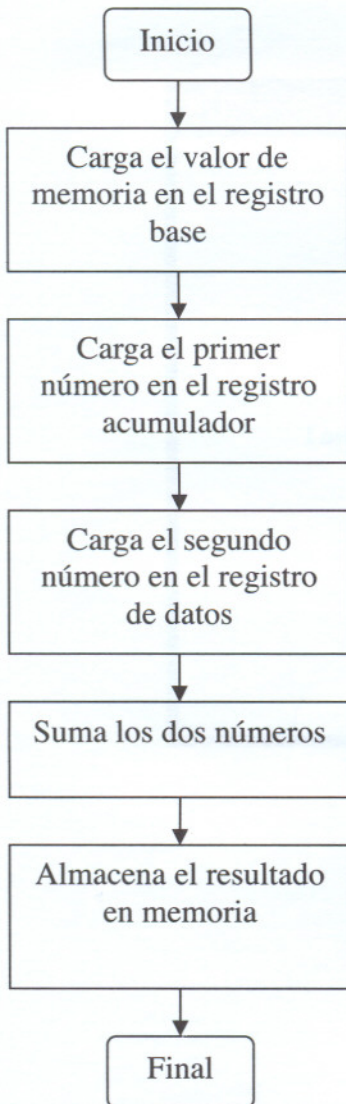
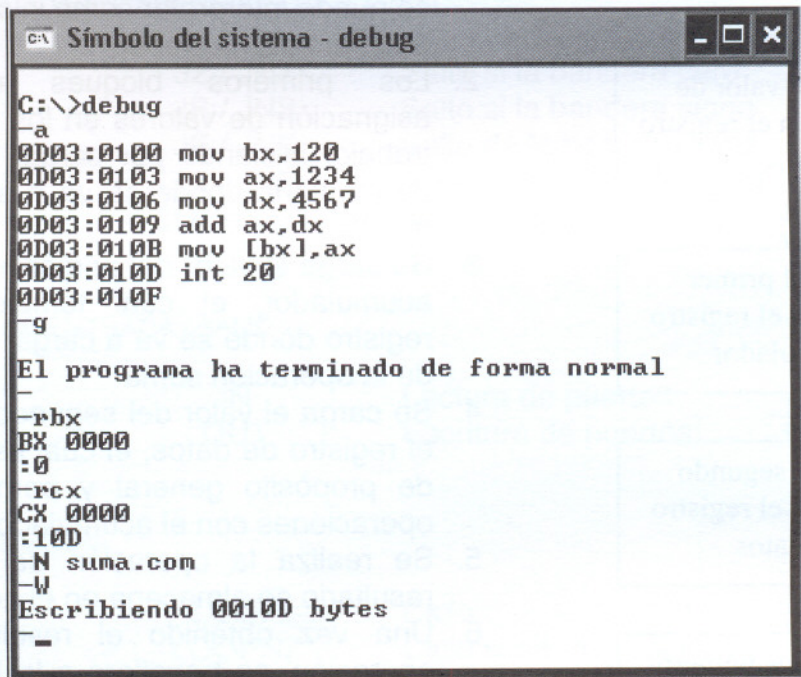
Carga el primer número en el registro acumulador

Carga el segundo número en el registro de datos

Suma los dos números

Almacena el resultado en memoria

Final

El algoritmo y la codificación es la siguiente:

1. El primer bloque establece el inicio, lo que se puede interpretar como iniciar la sesión en el programa Debug
2. Los primeros bloques marcan la asignación de valores en los registros de trabajo, iniciando por cargar la dirección de memoria donde se va a almacenar el resultado
3. Se carga el valor del primer número en el acumulador, el cual también será el registro donde se va a cargar el resultado de la operación suma
4. Se carga el valor del segundo número en el registro de datos, el cual es un registro de propósito general y permite realizar operaciones con el acumulador
5. Se realiza la operación de suma y el resultado se almacena en el acumulador
6. Una vez obtenido el resultado de la operación, se transfiere a la localidad de memoria
7. El programa se puede terminar con una llamada al sistema operativo, la cual es una función grabada en el BIOS.

```
CS:0100    mov bx, 120
CS:0103    mov ax, 1234
CS:0106    mov dx, 4567
CS:0109    add ax, dx
CS:010B    mov [bx], ax
CS:010D    int 20
```

Como resumen, el ejercicio anterior se codificaría siguiendo los siguientes pasos:

1. Se abre una sesión DOS
2. Se ejecuta la herramienta Debug desde la ruta que se quiera guardar el programa
3. Se codifican las instrucciones con el comando A
4. Se ejecuta el programa con el comando G.
5. Para guardar el código se siguen los siguientes pasos
   a. El tamaño del archivo se escribe en los registro BX:CX. En este caso, BX=0 y CX=10D
   b. Con el comando N se asigna el nombre del archivo, el cual será suma.com
   c. Con el comando W se graba el programa en la ruta seleccionada

Símbolo del sistema - debug

```
C:\>debug
-a
0D03:0100 mov bx,120
0D03:0103 mov ax,1234
0D03:0106 mov dx,4567
0D03:0109 add ax,dx
0D03:010B mov [bx],ax
0D03:010D int 20
0D03:010F
-g

El programa ha terminado de forma normal
-
-rbx
BX 0000
:0
-rcx
CX 0000
:10D
-N suma.com
-W
Escribiendo 0010D bytes
-_
```

Capítulo 8

Lenguaje C

Lenguaje C

DEFINICIÓN

La programación en lenguaje ensamblador es útil en algunas funciones de bajo nivel enfocadas al manejo directo del *hardware*. Por otro lado, C es un lenguaje de alto nivel que a menudo se utiliza en vez del lenguaje ensamblador para programar microprocesadores y aplicaciones avanzadas, ya que cuenta con un compilador más poderoso que simplifica el trabajo del programador y beneficia la interfaz con el usuario.

Hay muchas referencias que presentan a detalle el uso del lenguaje C, por lo que en este capítulo se pondrá énfasis en la forma en cómo combinar un programa C con rutinas ensamblador.

DISEÑO DE UN PROGRAMA

Decir que C es un lenguaje que cuenta con un compilador significa que hay dos etapas para generar una aplicación ejecutable. La primera consiste en escribir un código basado en comandos C y que representa la lógica diseñada en el algoritmo del programa y la segunda parte consiste en la traducción que se hace de ese código a lenguaje de máquina, lo que permitirá su posterior ejecución. Escribir un programa en C es algo más, por lo que conviene observar las recomendaciones de los siguientes siete pasos:

1. Análisis de los objetivos del programa
   Esta etapa es muy importante. En ella se establecen los datos de entrada, de salida, la forma en la que se van a procesar y la funcionalidad total del programa. Esta etapa no se tiene que asociar a ningún lenguaje de programación ni a comandos especiales. Entre mejor sea la definición de esta etapa, la codificación y pruebas del programa se facilitarán.

2. Diseño del algoritmo
   Una vez que se tiene un clara concepción del problema a resolver, la siguiente etapa es definir cómo se va a organizar el programa, cómo se van a representar los datos y qué comunicación va a tener con el usuario.

3. Codificación del algoritmo
   Esta etapa es la traducción del algoritmo en la secuencia necesaria de instrucciones usando exclusivamente comandos C y que generalmente se escriben en un archivo simple de texto al que se le llama archivo de código fuente.

4. Compilación del código
   El siguiente paso es compilar el código fuente, es decir, traducirlo a comandos que el microprocesador pueda entender y ejecutar. Durante este proceso, el compilador puede incluir algunas librerías externas para terminar el programa final. Estas librerías son rutinas estándar que se pueden usar. El compilador también verifica que la codificación sea correcta, por lo que en caso de haber errores, estos serán notificados al programador.

5. Ejecución del programa
   Una vez generado el archivo ejecutable, está listo para probarse. C es un compilador que genera archivos que se pueden ejecutar en diferentes ambientes, por lo que en la mayoría de los casos, pueden correr en diferentes sistemas operativos.

6. Pruebas y corrección de errores en el algoritmo
   Ahora que ya se tiene una aplicación ejecutable, hay que estar seguros de que manipula y ofrece la información correctamente. Pude suceder que el programa sí funciona pero los resultados son incorrectos.

7. Mantenimiento y actualizaciones
   La última etapa consiste en mantener una aplicación útil para el usuario. Una vez que se tiene un programa corriendo perfectamente, ahora hay que analizar nuevos cambios o nuevas adaptaciones, lo que nos llevaría nuevamente al primer paso.

ESTRUCTURA DE UN PROGRAMA

La programación C se ajusta a la metodología definida como programación estructurada que, en pocas palabras, define el formato de organización del código en el programa fuente. Esta metodología se resume en diseñar una codificación modular de las funciones del programa con una clara etapa preliminar de definición de variables. Las funciones del programa estarán coordinadas por una función principal llamada *main()*.

Esta estructura se puede representar en una forma gráfica y aclarar sus partes principales, según se muestra a continuación. La diferencia entre un programa simple y uno complejo radica en el número de funciones y la lógica del algoritmo, pero la estructura es igual en ambos casos.

Por lo tanto, es recomendable tomar en cuenta la estructura y apegarse siempre a ella, lo cual facilitará la codificación, sus pruebas y el mantenimiento posterior que se tenga que hacer del código.

Programa típico C

→ #include < > — Instrucciones de preproceso

→ main() — Función principal y primera en ser llamada

→ Declaraciones — Las funciones están formadas de declaraciones

→ función a()

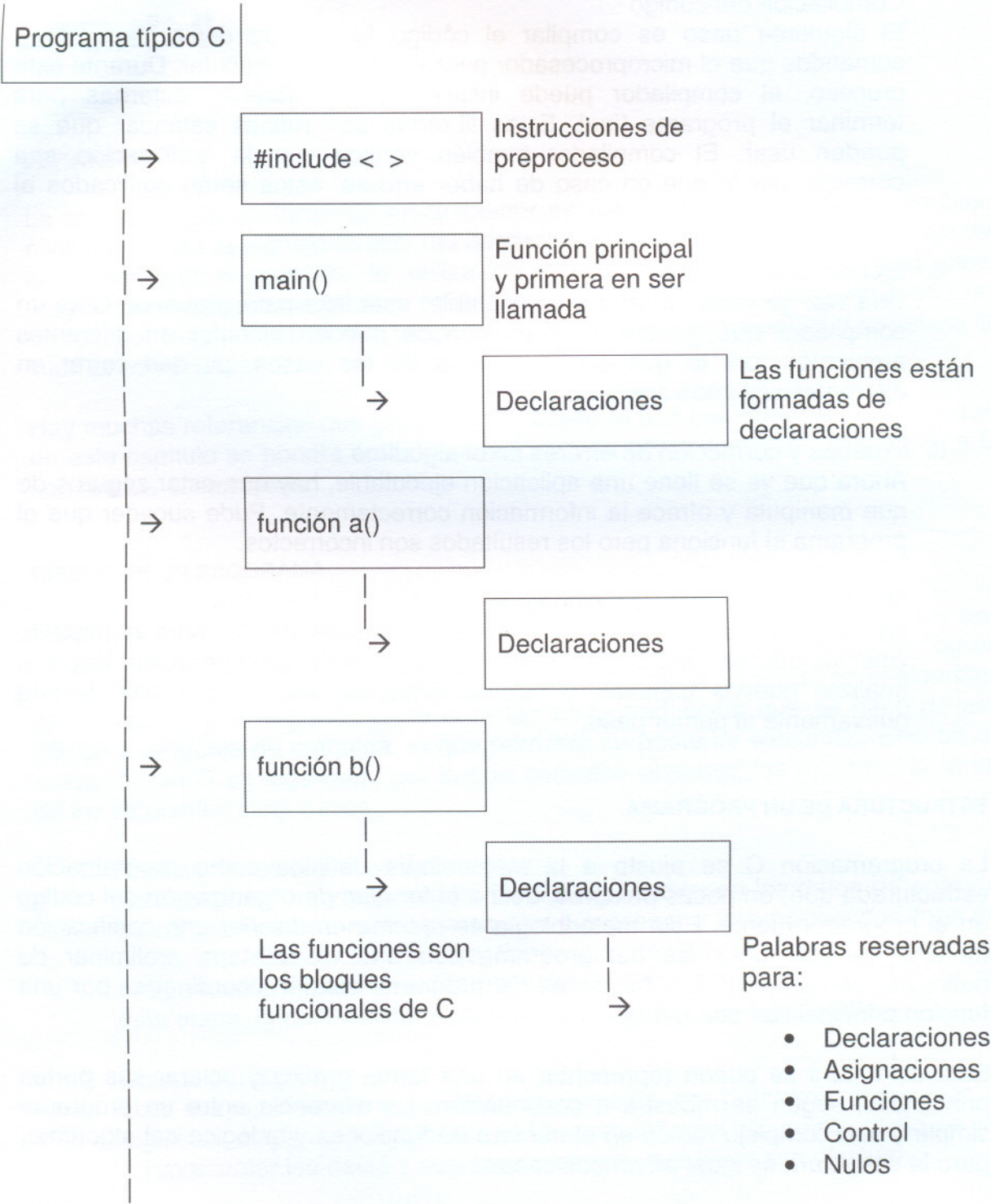
→ Declaraciones

→ función b()

→ Declaraciones

Las funciones son los bloques funcionales de C

→ Palabras reservadas para:

- Declaraciones
- Asignaciones
- Funciones
- Control
- Nulos

El siguiente es un ejemplo de cómo codificar un programa:

```
/* Codificación mínima de un programa */
#include <stdio.h>

main()
{
        int num;            /* definición de la variable num */

        num = 1;            /* asignación del valor a la variable */

        printf("Este es mi primer programa\n");          /* impresión del mensaje */
        printf("Mi número %d es mi favorito", num);

        return(0);
}
```

Aumentando una función en este simple programa quedaría de la siguiente forma:

```
/* Codificación mínima de un programa con una función*/
#include <stdio.h>

main()
{
        int num;            /* definición de la variable num */

        num = 1;            /* asignación del valor a la variable */

        printf ("Este es mi primer programa\n");          /* impresión del mensaje en pantalla */
        linea ();
        printf ("Mi número %d es mi favorito", num);

        return (0);
}

linea()
{
        printf ("Esta es la segunda línea\n");  /* impresión del mensaje en pantalla */

}
```

PALABRAS RESERVADAS

C tiene algunas palabras reservadas que usa como comandos para codificar los algoritmos. Cada librería que se incluye aumenta el número de palabras reservadas dentro de un código fuente.

El estándar ANSI C contiene las siguientes palabras

auto	break	case	char	const	continue
default	do	double	else	enum	extern
flota	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				

ESTRUCTURA DE DATOS

Las computadoras manipulan información. ¿Cómo se organizan los datos en lenguajes de alto nivel como C? La unidad básica de información en el *bit*, cuyos valores establecen una de dos posibilidades mutuamente excluyentes, "1" y "0". La agrupación de estos bits representa diferentes datos y se clasifican de la siguiente forma: A los diferentes grupos se les denomina byte (8 bits), palabra (16 bits) y palabra doble (32 bits)

NOMBRE		TAMAÑO	VALORES
char	caracter	1 byte	0 a 255
unsigned int	entero sin signo	2 bytes	0 a 65,535
int	entero con signo	2 bytes	-32,768 a 32,767
unsigned long	entero largo sin signo	4 bytes	0 a 4,294,967,295
long	entero largo con signo	4 bytes	-2,147,483,648 a 2,147,483,647
float con signo	coma flotante, real	4 bytes	-10^{38} a 10^{38}
float con signo	coma flotante largo	8 bytes	-10^{307} a 10^{307}

Función printf

Esta función permite imprimir información en la pantalla, tomándola como salida estándar. El formato es printf ("formato", argumentos). Ejemplo

printf ("Mi número %d es mi favorito", num)

Hay algunos caracteres indicadores de formato que permiten una determinada representación a la salida. Los principales son:

CARACTER	ARGUMENTOS	RESULTADO
d, i	entero	Entero decimal con signo
u	entero	Entero decimal sin signo
o	entero	Entero octal sin signo
X, x	entero	Entero hexadecimal sin signo
f	real	Real con punto y con signo
E, e	real	Notación exponencial con signo
c	caracter	Caracter
s	cadena de caracteres	Cadena de caracteres
ld, lu, lx, lo	entero	Entero largo

Hay algunas secuencias de escape que permite imprimir con formato el texto. Las principales son:

SECUENCIA	SIGNIFICADO
\n	Nueva línea
\t	Tabulador
\b	Espacio para atrás
\r	Retorno de carro
\"	Comillas

Variables

Una variable es un identificador usado para representar cierto valor. Cada variable es de un tipo de datos específico. Es una buena consideración que todas las variables empiecen con letra o con el caracter _. C es sensible a mayúsculas y minúsculas en la definición de variables.. Ejemplos

```
int num;
float _valor;
unsigned long entero;
```

Expresiones

Una expresión representa una unidad de datos simple. Puede estar formada por identificadores y operadores.

```
A + b
valor1 * valor2
```

Función scanf

Esta función permite leer datos de usuarios desde la entrada de datos estándar que es el teclado.. El formato es scanf("formato", argumentos). Ejemplo

```
scanf ( "%f", &numero);
```

Esta sentencia lee un número en coma flotante y lo almacena en la variable numero.

Operadores aritméticos

Los operadores aritméticos en C son

OPERADOR	FUNCIÓN
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo, resto de la división entera

Operadores relacionales

Los operadores relaciones en C son

OPERADOR	FUNCIÓN
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual que
¡=	Distinto

Operadores lógicos

Los operadores lógicos en C son

OPERADOR	FUNCIÓN
&&	AND
ll	OR
¡	NOT

Operadores de asignación

La forma general del operador de asignación es

identificador = expresión

El operador de asignación = y el de igualdad == son diferentes. La asignación se efectúa de derecha a izquierda

Operadores de asignación compuestos

C permite la utilización de los operadores compuestos, cuyo significado es

EXPRESIÓN	EXPRESIÓN EQUIVALENTE
J += 6	J = J + 6
J -= 6	J = J – 6
J *= 6	J = J * 6
J /= 6	J = J / 6
J %= 6	J = J % 6

SENTENCIAS DE CONTROL

C ofrece varias sentencias de control, las cuales van a permitir controlar el flujo del programa según la evaluación de sus condiciones. Los formatos básicos son los siguientes:

Sentencia if

```
if (expresión)
        sentencia;
```

Si expresión es verdadera se ejecuta sentencia. Expresión debe estar entre paréntesis. En caso de que sentencia sea compuesta, entonces

```
if (expresión)
        {
        sentencia 1
        sentencia 2
        :
        :
        sentencia N
        }
```

Sentencia if - else

```
if (expresión)
        sentencia 1;
else
        sentencia 2;
```

Si expresión es verdadera se ejecuta sentencia 1. En el caso de ser falsa se ejecuta sentencia 2. Si las sentencias son compuestas se encierran entre { }.

Sentencia for

```
for (expresión 1; expresión 2; expresión 3)
        sentencia;
```

Inicialmente se ejecuta expresión 1. Esta expresión se inicializa con algún parámetro que controla la repetición del ciclo. La expresión 2 es una condición que debe ser cierta para que se ejecute sentencia. La expresión 3 se utiliza para modificar los valores del parámetro de la expresión 1. Ejemplo

```
main ()
{
        int valor;
        for (valor=0; valor <100; valor++)
                printf("%d\n", valor);
}
```

Sentencia while

```
while (expresión)
        sentencia;
```

Sentencia se ejecutará mientras el valor de expresión se verdadero, la cual es evaluada en primer lugar. Lo normal es que sentencia tenga algún elemento que modifique el valor de expresión, lo que implica que sentencia es compuesta.

```
while (expresión)
{
        sentencia 1
        sentencia 2
        :
        :
        sentencia N
}
```

Sentencia do-while

```
do
        sentencia
while (expresión);
```

Sentencia se ejecuta mientras el valor de expresión sea verdadero. En este tipo de ciclos, sentencia se ejecuta por lo menos una vez. Lo normal es que sentencia incluya algún elemento que modifique el valor de expresión, lo que implica que sentencia es compuesta

```
do
{
        sentencia 1
        sentencia 2
        :
        :
        sentencia N
} while (expresión);
```

Sentencia switch

```
Switch (expresión)
{
        case exp 1:
                sentencia 1;
                sentencia 2:
                :
                :
                sentencia N
                break;
        case exp 2:
                sentencia 21;
                sentencia 22:
```

```
                :
                :
                sentencia 2N
                break;
        default:
                sentencia d1;
                sentencia d2:
                :
                :
                sentencia dN
}
```

Expresión devuelve un valor entero y las expresiones exp 1 y exp 2 representan valores enteros iguales a las que devuelve expresión.

FUNCIONES

Una Función es un segmento de programa que realiza una determinada tarea. Todo programa C consta de una o varias funciones, donde main () es la principal. El uso de funciones permite descomponer en módulos la programación.

El formato general de la función es

```
tipo nombre (tipo1 arg1, tipo2 arg2, ... tipoN argN)
{
        /* Código de la función */

        return (expresión);
}
```

Los argumentos se denominan parámetros formales y el tipo de la función se refiere al tipo de dato que regresa. La sentencia *return* finaliza la ejecución y devuelve el valor a la función que realizó la llamada.

Si la función no requiere parámetros y no regresa ningún tipo de dato se pone *void.*

```
void nombre (void)
{
        /* Código de la función */
}
```

Respecto al uso de funciones hay que tomar en cuenta que una función C sólo puede devolver un valor y que no es posible anidar funciones.

Declaración de prototipos

La forma general de declara un prototipo es

```
tipo nombre (tipo1 arg1, tipo2 arg2, ... tipoN argN);
```

La declaración o prototipo de una función especifica el nombre de la función, el valor que devuelve y los tipos de parámetros que acepta. Esta declaración termina con ; y no incluye el cuerpo de la función.

```
tipo nombre (tipo1, tipo2, ... tipoN) /* Prototipo */

tipo nombre (tipo1 arg1, tipo2 arg2, ... tipoN argN) /* Definición */
{
        /* Código de la función */

        return (expresión);
}
```

Llamadas a funciones

Para llamar a una función se especifica el nombre y la lista de parámetros. Por ejemplo

```
suma ( 2 , 4 ):
```

donde los parámetros formales deben ser del tipo definido en el prototipo de la función.

INLINE ASSEMBLER

El lenguaje ensamblador sirve para muchos propósitos entre los que se puede mencionar el aumento de velocidad en la ejecución de una función, la disminución en el uso de memoria ya que una rutina bien desarrollada no necesita espacio adicional para validaciones innecesarias y el control que se tiene sobre el *hardware* del sistema.

El ensamblador embebido en el código C es la opción para contar con estos beneficios y es algo fácil de implementar ya que el compilador ensamblador está incluido en el compilador C de la mayoría de las marcas y versiones.

El único cuidado que hay que tener es que los programas podrían no ser portables al 100% ya que el código ensamblador está destinado a un tipo de *hardware* o de

microprocesador en especial, Las siguientes consideraciones también son importantes:

- El compilador C no va a tratar de optimizar el código ensamblador, por lo tanto no habrá corrección de errores.
- Los bloques de código ensamblador pueden modificar los valores de los registros del microprocesador, por lo que el control queda en manos del programador
- Se limita toda opción de optimización del código fuente en general

_asm (Palabra reservada)

La palabra _asm es un comando reservado en varias versiones de compiladores C. Este comando permite la ejecución en línea de código ensamblador y puede ser usado en cualquier parte del código fuente C. Este código ensamblador debe estar entre { } precedido del comando _asm.

El formato del comando es

```
_asm {
        mnemónico (destino, origen)
        :
        :
        mnemónico (destino, origen)
        }
```

En caso de que el código ensamblador sea una sola línea, se pueden evitar los { }

Como primer ejemplo de una función, se presenta el siguiente código

```
void pantalla ( void )
{
        _asm
            {
                mov ah,00
                mov al,03
                int 10
            }
}
```

Esta función configura en modo de texto de 80 x 25 la pantalla DOS y está escrita con el formalismo apropiado.

Este código también se podría escribir de la siguiente forma

```
_asm mov ah,00
_asm mov al,03
_asm int 10
```

Este código muestra que no es necesario terminar el renglón con **;** ya que _asm es un comando que incluye un separador, por lo que también se podría escribir

```
_asm mov ah,00    _asm mov al,03    _asm int 10
```

Etiquetas

Como en cualquier compilador C, el código ensamblador se auxilia de las etiquetas para controlar el flujo del programa.

Las etiquetas se representan con su nombre seguido de dos puntos

```
nombre_de_etiqueta :
```

Las etiquetas no son sensibles a mayúsculas o minúsculas y los diferentes compiladores tienen dos formas de tratarlas: dentro de los corchetes del comando _asm o fuera de los corchetes. Ejemplo

```
void pantalla ( void )
{

        _asm
                {
                        mov ah,f
                }
        inicio:
        _asm
                {
                        dec ah
                        jnz inicio
                        int 20
                }
}
```

Esta rutina asigna al registro ah el valor f y lo decrementa hasta llegar a cero. Para formar el ciclo, se usa la etiqueta inicio para volver a evaluar el resultado.

Capítulo 9
Lenguajes visuales

Lenguajes visuales

EL ENTORNO DE PROGRAMACIÓN

Los lenguajes de programación son herramientas que mediante una secuencia de comandos, nos permiten codificar instrucciones de manera que sean entendidas y ejecutadas por el microprocesador de una computadora.

Hay varias metodologías para comunicar esta lista de comando al microprocesador. Un intérprete es aquel lenguaje que no trabaja en código máquina en forma directa, sino que va traduciendo cada instrucción. Estos lenguajes son mucho más lentos que los lenguajes de alto nivel que trabajan ejecutando instrucciones directamente en código máquina. Un ejemplo muy popular de este tipo de lenguajes es el QBASIC.

Los lenguajes que ejecutan las instrucciones en lenguaje máquina tienen un traductor llamado compilador. Un compilador traduce las instrucciones del lenguaje contenidas en el código fuente (instrucciones) a código máquina, de manera que el programa no necesita interpretar o convertir cada instrucción. Debido a esto es mucho más veloz que un intérprete y por supuesto mucho más eficiente. La calidad del programa no depende de esta eficiencia, ya que tanto un método como el otro pueden tener algoritmos de calidad.

A partir de la aparición de C, el gran lenguaje, y Pascal, se dividen los lenguajes en estructurados y no estructurados. Los lenguajes estructurados son aquellos que en su codificación usan una estructura jerárquica de procedimientos y funciones, mientras que los lenguajes no estructurados, como el Basic, usan una codificación que se basa en líneas de programación, permitiendo al programador "saltar" de una línea de instrucción a otra, haciendo que el código sea algunas veces difícil de entender y muy difícil de mantener.

Con el tiempo surge una variante que se aplica a todos los lenguajes: La orientación a objetos. Ya no solo se habla de programación estructurada, sino que los módulos de programación son vistos como objetos, las estructuras representan objetos y/o funciones que se adaptan en forma general a procesos específicos, haciendo que la programación modular alcance un gran desarrollo.

El modelo de objetos engloba los conceptos de encapsulación, herencia y poliformismo, los cuales se aplican a los datos y al tipo de bases de datos que almacena la información.

La orientación a objetos significa la agrupación de entidades de datos de forma global, de tal manera que puedan ser interpretados de una forma común por una misma estructura de programación.

Actualmente, con la sólida evolución que han tenido los sistemas operativos visuales, todo es visual, todo es iconos, todo es botones, todo es ventanas. Por lo tanto, para programar en lenguajes visuales, primero hay que comprender lo que son estos sistemas operativos, del que Windows puede ser una marca muy comercial.

La forma de programar los sistemas evolucionó radicalmente. Con Windows es preciso programar conservando las convenciones del mismo, guardando sus características y funcionalidades.

La forma de programar se basa en objetos, cada uno de los cuales tiene sus propiedades y funciones. Se basa en la programación de eventos para dichos objetos. Otro detalle es que la programación se basa en componentes denominados OLE, OCX y ActiveX, los cuales reducen notablemente el trabajo de la programación al proporcionar herramientas antes impensadas en la programación D.O.S.

Todos los lenguajes visuales ofrecen RAD (Rapid Apl ication Development) o Wizards, con lo cual comenzaron a prometer hacer aplicaciones en poco tiempo, incluso para inexpertos. Ahora las herramientas de programación son poderosas. Son casi un sistema operativo con entornos de desarrollo avanzados y excelentes Debuggers.

Los paradigmas de la programación Windows, entre otros, son:

- Borland Delphi (la evolución del Pascal)
- Visual Fox (la evolución del Xbase)
- Visual Basic (la evolución de Basic)
- Visual C++ (la evolución del C/C++)

Las incursiones cada vez mas innovadoras de Microsoft parecen imponer a Internet como el centro de desarrollo de aplicaciones .NET. Las nuevas tecnologías WEB inundan el mercado: PHP, ASP, XML, DHTML, lo cual enriquecen la forma de manejar la información y su presentación al usuario final.

Es de esperarse que los lenguajes Visuales dominen el mundo de las PC's durante mucho tiempo, por lo menos mientras no evolucionen de otra manera los Sistemas Operativos. Los lenguajes de programación evolucionan a medida que lo hacen los Sistemas Operativos en que funcionan. Nunca un lenguaje de programación determinó un Sistema Operativo, por el contrario los Sistemas Operativos determinaron los lenguajes de programación.

MS VISUAL BASIC

Visual Basic ha sido una plataforma importante de desarrollo ya que ofrece una fácil programación de aplicaciones. Visual Basic fue diseñado para crear aplicaciones en sistemas operativos de 32 bits, por lo que es una herramienta desde la aparición de Windows95. Estas aplicaciones de 32 bits son muy eficientes debido al espacio de direcciones que pueden manejar en memoria, por lo que las aplicaciones aprovechan los recursos que ofrece el sistema operativo siendo más robustas y con buenas capacidades de trabajo en entornos multitareas y multiprocesos, superando las antiguas aplicaciones de 16 bits.

Se pueden concluir las siguientes ventajas:

- Acceso al espacio completo de direcciones de 32 bits de memoria con microprocesadores 386 y superior alcanzando hasta 4 GBytes
- Funcionamiento mejorado gracias a los cálculos y operaciones de memoria realizados con 32 bits
- Mayor protección contra fallas del sistema como resultado de acciones inapropiadas por parte de otros programas, ya que cada aplicación Visual Basic usa una región protegida de memoria.
- Capacidades reales de multitarea, ya que la ejecución del programa se puede detener en cualquier momento y desarrollar otras funciones.
- Acceso a la API Win32 del sistema operativo, teniendo acceso a funciones más avanzadas de programación

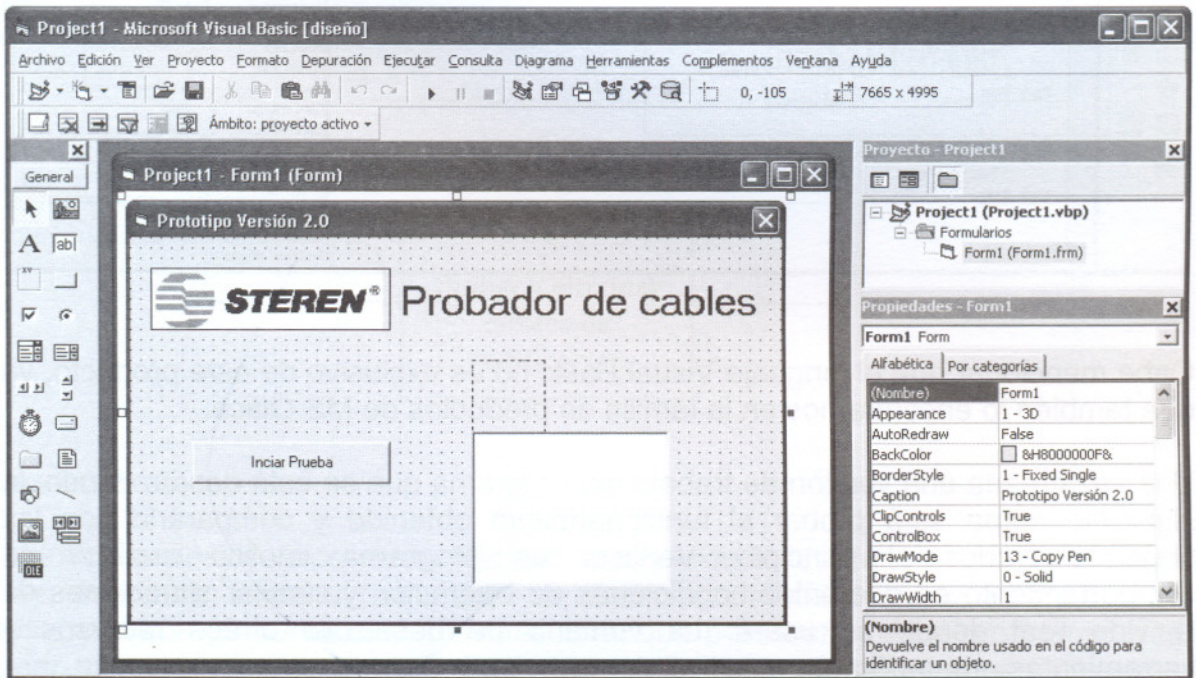
Para escribir un programa en Visual Basic se deberá determinar, en primer lugar, cuáles serán las tareas que deberá desarrollar; posteriormente, se deberá diseñar la pantalla o interfaz del usuario, la cual contendrá diferentes componentes u objetos que servirán para la interacción con el usuario y desarrollarán las funciones necesarias para el procesamiento de datos; y por último se hará la programación de los objetos y del sistema en general.

Por lo tanto, desarrollar una aplicación para Windows usando Visual Basic suele implicar tres pasos generales:

1. Creación de la interfaz del usuario
2. Definición de las características o propiedades de los elementos presentes en la interfaz del usuario
3. Escribir el código asociado a cada uno de los elementos de la interfaz del usuario

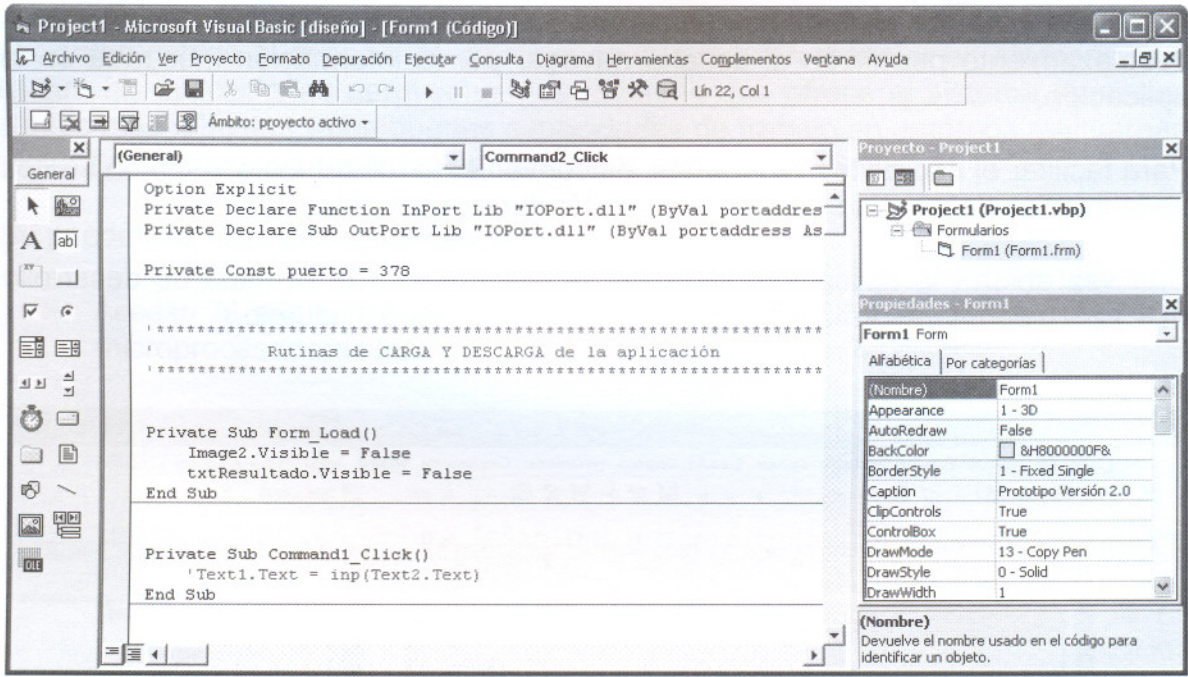
La interfaz del usuario incluirá todos los menús, cuadros de diálogo, botones, objetos y dibujos que el usuario utilizará para trabajar con la aplicación. Los menús contendrán todas las opciones disponibles en la aplicación; los cuadros de diálogo, botones y el cursor del *mouse* ayudarán al usuario a manipular la aplicación y la información que se estará procesando; las ventanas y las barras de desplazamiento permitirán al usuario navegar por la información mostrada en la aplicación.

Para facilitar el uso de los programas, es conveniente utilizar los objetos presentes en la interfaz del usuario de una forma estándar y predecible.

Una vez abierta la aplicación, se podrá tener acceso a la ventana de desarrollo (IDE) para definir la forma en la que se verá la interfaz del usuario.

Para construir los elementos de la interfaz, bastará con seleccionar y arrastrar a la ventana diseñada para la aplicación (forma) los objetos deseados de las barras a la izquierda de los cuadros de herramientas. Se trata de una labor realmente sencilla.

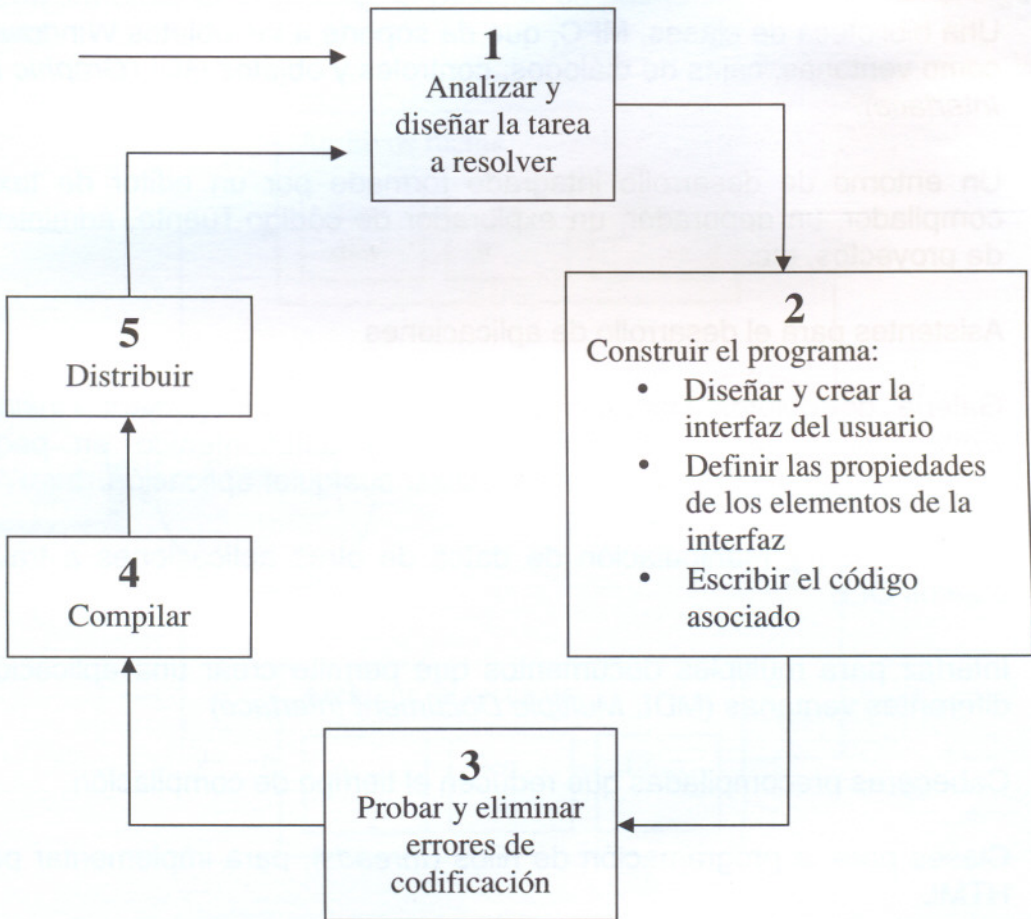
Una vez creado el elemento se podrán definir las propiedades asociadas con él y posteriormente introduciendo el código que desarrollará las funciones del elemento. Esta codificación se hará en la ventana de código.

El lenguaje de programación Visual Basic cuenta con cientos de instrucciones, funciones y caracteres especiales, aunque la ventana de codificación tiene grandes ayudas para facilitar y recordar la escritura de las instrucciones.

```
Option Explicit
Private Declare Function InPort Lib "IOPort.dll" (ByVal portaddres
Private Declare Sub OutPort Lib "IOPort.dll" (ByVal portaddress As

Private Const puerto = 378

'******************************************************************
'          Rutinas de CARGA Y DESCARGA de la aplicación
'******************************************************************

Private Sub Form_Load()
    Image2.Visible = False
    txtResultado.Visible = False
End Sub

Private Sub Command1_Click()
    'Text1.Text = inp(Text2.Text)
End Sub
```

Cabe mencionar que el lenguaje Visual Basic no es exclusivo de este producto, ya que también lo encontramos en la familia de productos de MS Office.

Una vez creada una versión de trabajo del programa que se está desarrollando, la siguiente etapa será probar el funcionamiento obtenido y compararlo con los objetivos fijados al principio. Verificar un programa implica analizar su comportamiento en diferentes condiciones de operación y simular situaciones de la vida real donde se usará. La ventana de desarrollo ofrece recursos y herramientas que facilitarán este trabajo.

Si se desea distribuir el programa, que generalmente será una de las intenciones del desarrollo de aplicaciones, se necesitará compilarlo para generar un archivo ejecutable y una pequeña herramienta de instalación en otras computadoras. Todos los programas desarrollados en Visual Basic requieren uno o más archivos denominados Librerías de Enlace Dinámico, DLL, para poder ejecutarse correctamente, archivos que se incluirán en la versión de distribución.

Estos pasos reseñan el ciclo total en la vida de desarrollo de software, el cual se puede ilustrar de la siguiente forma:

MS VISUAL C++

Cuando los desarrolladores de Microsoft planearon la idea que respalda a Visual C++, decidieron tomar el compilador de clase mundial C++ y crear un entorno de desarrollo con un conjunto de herramientas que permitieran a los programadores crear aplicaciones Windows con un nivel de facilidad y rapidez superior a otros entornos de desarrollo.

Con esta herramienta, se introdujo la tecnología de desarrollo a base de asistentes con una nueva versión de las MFC (Microsoft *Foundation Class Library*) más potente, lo cual facilita enormemente el desarrollo de aplicaciones.

Al igual que Visual Basic y otras herramientas visuales orientadas a objetos, Visual C++ es un entorno de desarrollo diseñado especialmente para crear aplicaciones gráficas orientadas a objetos sobre una plataforma de 32 bits. Para crear una aplicación se crean ventanas para acomodar controles programables según las funciones deseadas por el programador.

En resumen, Visual C++ en una forma general, tiene las siguientes características

- Una biblioteca de clases, MFC, que da soporte a los objetos Windows tales como ventanas, cajas de diálogos, controles y objetos GDI (*Graphic Device Interface*).

- Un entorno de desarrollo integrado formado por un editor de texto, un compilador, un depurador, un explorador de código fuente, administración de proyectos, etc.

- Asistentes para el desarrollo de aplicaciones

- Galería de objetos incrustados y vinculados OLE (*Object Linking and Embedding*). Esto se refiere a software autocontenido en pequeñas unidades de software que puede reutilizar cualquier aplicación.

- Visualización y manipulación de datos de otras aplicaciones a través de objetos OLE

- Interfaz para múltiples documentos que permite crear una aplicación con diferentes ventanas (MDI, *Multiple Document Interface*)

- Cabeceras precompiladas que reducen el tiempo de compilación

- Clases para la programación de hilos (*threads*), para implementar páginas HTML

- Creación y uso de librerías dinámicas (DLL, *Dinamic Link Libraries*)

- Programación Internet a través de componentes *Active-X*

- Soporte para el estándar COM (*Component Object Model*)

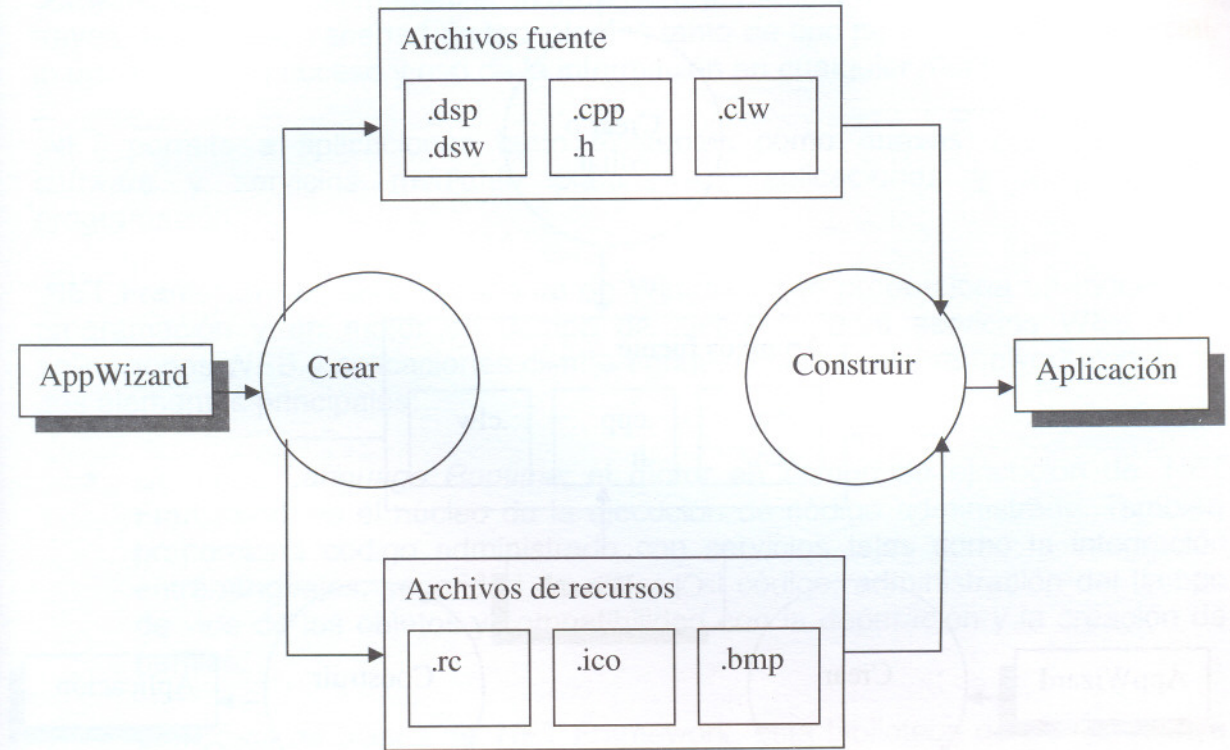
- Objetos de acceso a datos DAO, que permita el acceso a bases de datos a través del motor Access o de controladores ODBC

- OLE DB como un proveedor de datos y objetos ADO (*Active-X Data Objects*)

Creación de una aplicación

El proceso de desarrollo de una aplicación Visual C++ se puede dividir en dos fases generales: creación del esqueleto de la aplicación y el desarrollo de la aplicación.

Cuando se crea una aplicación, el primer paso es ejecutar el asistente *AppWizard* que permitirá crear el esqueleto. El proceso se puede representar con el siguiente diagrama

Archivos fuente
.dsp
.dsw
.cpp
.h
.clw
AppWizard
Crear
Construir
Aplicación
Archivos de recursos
.rc
.ico
.bmp

La fase de desarrollo incluye el diseño de la interfaz del usuario, la edición de archivos fuente, compilar la aplicación, el enlace, la prueba y la depuración final.

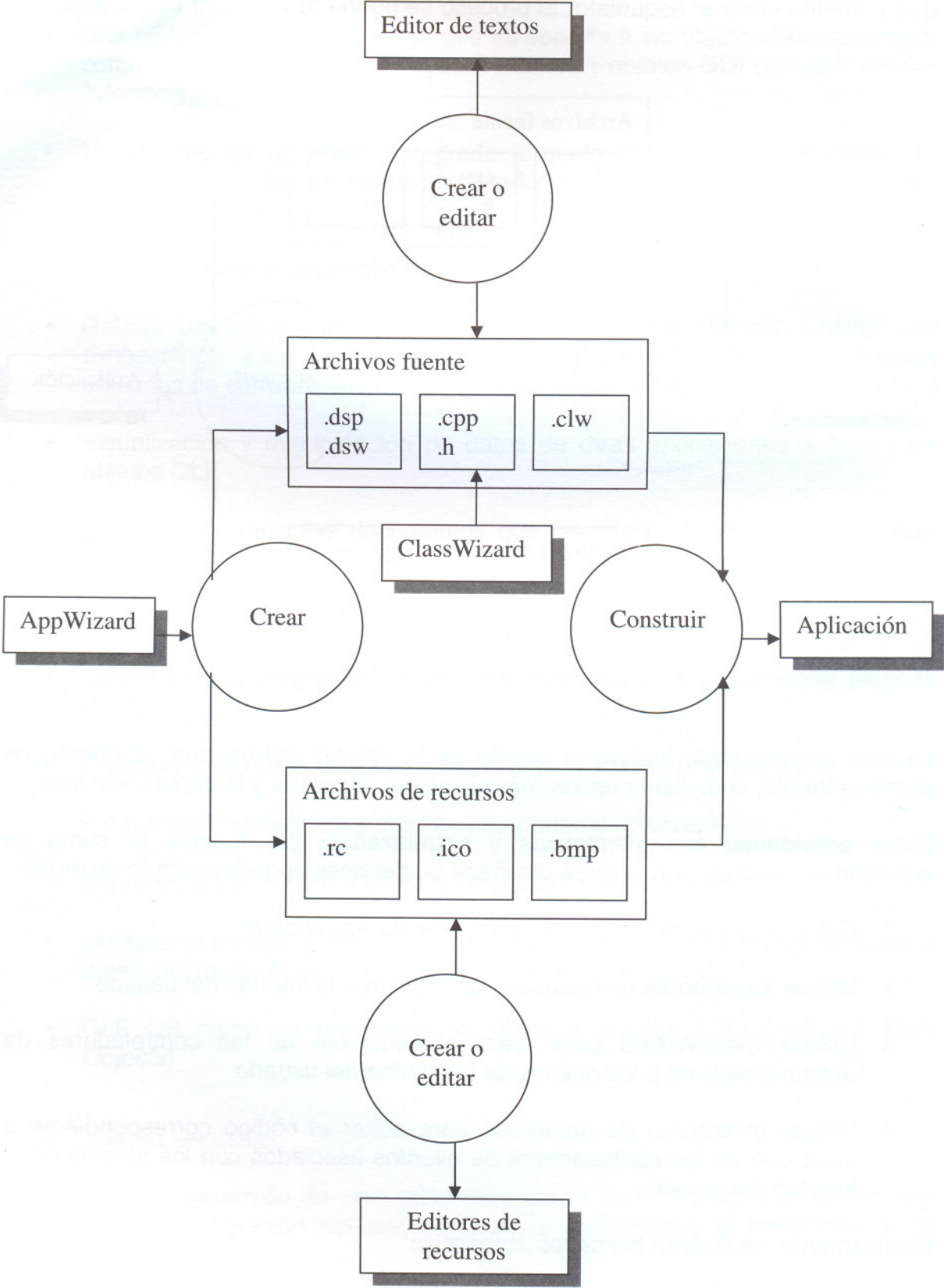
Estas actividades son interactivas y entrelazadas, por lo que la suma de actividades concluye con la tarea diseñada pudiéndose sugerir el siguiente orden:

1. Crear el esqueleto de la aplicación usando AppWizard

2. Utilizar los editores de recursos para construir la interfaz del usuario

3. Utilizar ClassWizard para crear el esqueleto de los controladores de eventos relativos a los objetos de la interfaz del usuario

4. Utilizar el entorno de desarrollo para editar el código correspondiente a cada uno de los controladores de eventos asociados con los objetos de la interfaz del usuario.

Gráficamente, se ilustran los pasos anteriores:

TECNOLOGÍA .NET

Microsoft .NET es una nueva plataforma no solo de programación sino de tecnología en general. Microsoft define que la tecnología .NET es un tipo de software capaz de interconectar información, personas, sistemas y dispositivos a través de unir una variedad de tecnologías tanto de tipo personal como comercial, lo que permite el acceso y uso de la información en cualquier momento y lugar.

.NET permite a aplicaciones tanto existentes como nuevas, conectarse con software y servicios mediante plataformas, aplicaciones y lenguajes de programación.

.NET Framework es un componente de Windows que proporciona un modelo de programación y un motor en tiempo de ejecución para servicios WEB XML, aplicaciones WEB y aplicaciones cliente enriquecidas. .NET Framework consta de dos elementos principales:

- *Common Language Runtime*: el motor en tiempo de ejecución de .NET Framework es el núcleo de la ejecución de código administrado. También proporciona código administrado con servicios tales como la integración entre lenguajes, seguridad de acceso al código, administración del tiempo de vida de los objetos y compatibilidad con la depuración y la creación de perfiles.

- *Biblioteca de clases de* .NET Framework: esta biblioteca de clases incluye clase, interfaces y tipo de valores que expiden y optimizan el proceso de desarrollo proporcionando acceso a las funciones del sistema.

Visual Studio .NET es la herramienta para crear servicios WEB XML y aplicaciones cliente .NET. Proporciona un entorno de desarrollo muy completo para la creación de aplicaciones escalabes seguras en el lenguaje que se quiera, aprovechando los sistemas y conocimientos disponibles, ya que el Framework es común al lenguaje del programador que se quiera usar.

.NET Framework está diseñado para cumplir los siguientes objetivos:

- Proporcionar un entorno sólido de ejecución orientado a objetos, ya sea que el código de objetos se almacene y se ejecute localmente, se ejecute en forma remota o se ejecute en forma local pero se distribuya por Internet

- Proporcionar un entorno de ejecución de código que minimice la implementación de software y los conflictos entre versiones

- Proporcionar un entorno de ejecución de código que garantice la seguridad de la ejecución del código, incluido el código creado por un tercero desconocido

- Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos de secuencia de comandos o de interpretación

- Proporciona a los programadores una experiencia sólida en los diferentes tipos de aplicaciones, ya sean aplicaciones basadas en Windows o en WEB

- Basar todos los sistemas de comunicación, incluidos los servicios WEB XML y la conexión remota de punto a punto, en el estándar del sector para garantizar que el código basado en .NET Framework se pueda integrar con cualquier otro tipo de código, con independencia del lenguaje o de la plataforma en los que se haya escrito.

La biblioteca de clases de .NET Framework es una colección de tipos reutilizables que se integran de forma compacta en el *Common Language Runtime*. La biblioteca de clases está orientada a objetos, por lo que proporciona tipos desde los que su propio código administrado puede derivar funcionalidad. Esto no solo facilita el uso de los tipos de .NET Framework, sino que también reduce el tiempo asociado con el aprendizaje de nuevas características de :NET Framework. Además, existe una gran variedad de componentes de otros fabricantes que se pueden integrar con las clases en .NET Framework.

Con una sólida arquitectura orientada a objetos, .NET Framework se puede ampliar sin limitaciones, permitiéndole mejorar, modificar y ampliar sus funcionalidades básicas para adecuarse a las necesidades del usuario o a las empresas. Por ejemplo, la colección de clases de .NET Framework implementa un conjunto de interfaces que se pueden utilizar para desarrollar sus propias colecciones, que se integrarán sin ningún problema con otros objetos creados por otros usuarios o pertenecientes al sistema.

.NET Framework permite conseguir una variedad de tareas de programación comunes, como son la administración de cadenas, la recopilación de datos, la conectividad con bases de datos y el acceso a archivos.

Los desarrolladores de aplicaciones para Windows comprobarán que los formularios Windows Forms son intuitivos, eficientes y compatibles con cualquier lenguaje basado en .NET incluyendo Visual Basic, C#, y las extensiones administradas para C++.

Los formularios Windows Forms se beneficiarán del control de versiones y de las características de implementación de .NET Framwork para ofrecer costos de implementación reducidos y una mayor estabilidad de las aplicaciones.

Como ejemplo y compatibilidad visual con versiones anteriores, las siguientes imágenes representan el ambiente de desarrollo para Visual Basic .NET

WindowsApplication3 - Microsoft Visual Basic .NET [diseñar] - Form1.vb [Diseño]

Form1

Puerto Dato

In Out

WindowsApplication3 - Microsoft Visual Basic .NET [diseñar] - Form1.vb

```
Option Explicit On

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private Declare Auto Function InPort Lib "IOPort.dll"
    Private Declare Auto Sub OutPort Lib "IOPort.dll" (By'

 Código generado por el Diseñador de Windows Forms

    Private Sub Form1_Load(ByVal sender As System.Object,

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Obje
        TextBox2.Text = InPort("&h" + TextBox1.Text)
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Obje
        OutPort(Val("&h" + TextBox1.Text), TextBox2.Text)
    End Sub
End Class
```

Capítulo 10

Interfaz GPIO modelo K-400

Interfaz GPIO modelo K-400

¿POR QUÉ UNA INTERFAZ DE 8 BITS?

Hablando en confianza, la tecnología ha llenado nuestras actividades del día a día con muchos servicios y productos que nos permiten desarrollarlas de muchas y mejores maneras cada una de ellas. De hecho, haciendo una encuesta entre las personas del círculo social que nos rodea, veremos que somos privilegiados ya que formamos parte de una sociedad altamente tecnificada.

¿A qué me refiero? A la cantidad de dispositivos tecnológicos con los que contamos. Como primer ejemplo puedo mencionar el teléfono celular, el cual cuenta con cámara y videocámara digital, reproductor de música, agenda electrónica, calculadora, conexión a Internet y un sinnúmero de servicios adicionales. También contamos con computadoras portátiles y de escritorio, memorias portátiles que nos permiten transportar GBytes de información, servicios de televisión por cable, horno de microondas, etc., etc., etc.

Todos estos productos y servicios, sin duda, nos han hecho excelentes usuarios de la tecnología. Ahora lo único que tenemos que pedir es que no se interrumpa el suministro de energía eléctrica en las centrales de servicio para no vernos afectados.

Muy bien, ahora hay que tomar un poco de tiempo para analizar cual es una característica común entre todos estos dispositivos y que nos interesa considerar para empezar la reflexión sobre el tema de por qué una interfaz de 8 bits: pues lo común es que todos ellos tienen un diseño electrónico determinando y un software que se encarga de generar las diferentes funcionalidades.

Y este es el punto. Ya no hay diferencia entre hardware y software y ahora son una sola pieza que funciona en coordinación para desarrollar todas las funciones diseñadas en un producto y que cuando hay mejoras en ellos, basta con actualizar el software para contar con las nuevas funciones.

Como estoy seguro que esto es cierto y no hay forma de ponerlo en duda, la pregunta que sigue es ¿Y que hacemos ahora? En la parte de la vida diaria, nada, seguir como hasta ahora consumiendo productos y servicios tecnológicos. En la parte académica, escolar, formar a nuestros ingenieros en este sentido, en mostrar que la división entre electrónica y computación se ha borrado y ahora hay que saber y saber juntar los dos temas.

Por eso surge este proyecto, para poder entender las bases de esta importante relación -electrónica y computación- y poder generar los primeros experimentos.

Esta interfaz permitirá a los estudiantes de ingeniería y a los interesados en el tema a desarrollar los primeros prototipos electrónicos y controlarlos a través de la interfaz con programas desarrollados en diferentes lenguajes.

Para lograr esta comunicación entre la computadora y el prototipo electrónico, a través de la interfaz, es necesario usar rutinas escritas en lenguaje ensamblador que posteriormente se integrarán en lenguajes de alto nivel como C/C++, o cualquier otro lenguaje visual tipo Visual Basic o Visual C++.

Queda la libertad a la creatividad, ya que este capitulo presenta y explica las rutinas básicas de programación de la interfaz y su uso en unión a algunos proyectos sugeridos, con lo que posteriormente cada persona podrá desarrollar sus propios proyectos.

EL PUERTO PARALELO

Sin mayor preámbulo, empezaré explicando los conceptos necesarios para poder trabajar con el puerto paralelo de la computadora, el cual será el canal de comunicación con la interfaz y con el proyecto electrónico a controlar.

Sólo para aclarar, ¿por qué el puerto paralelo? Desde luego que tenemos otras opciones, como sería el puerto serial o el puerto USB pero al usar cualquiera de estas, también aumentaría la complejidad en la comunicación y para empezar, ya tenemos suficiente con nuestro planteamiento como para aumentar un problema más en la comunicación. Por lo tanto, este proyecto de interfaz no se complica y maneja la forma de comunicación más fácil. Una vez que se tenga experiencia en la elaboración de proyectos, entonces se podrá complicar un poco modificando el formato de comunicación.

Descripción de puerto

El modelo más simple pero útil que podemos hacer de una computadora es el que considera tres bloques básicos: procesador, memoria y puertos. Este modelo sugerido por John von Newmann sigue siendo, después de muchos años, un modelo válido y totalmente usado en la actualidad.

Los siguientes bloques presentan el concepto modular:

E/S

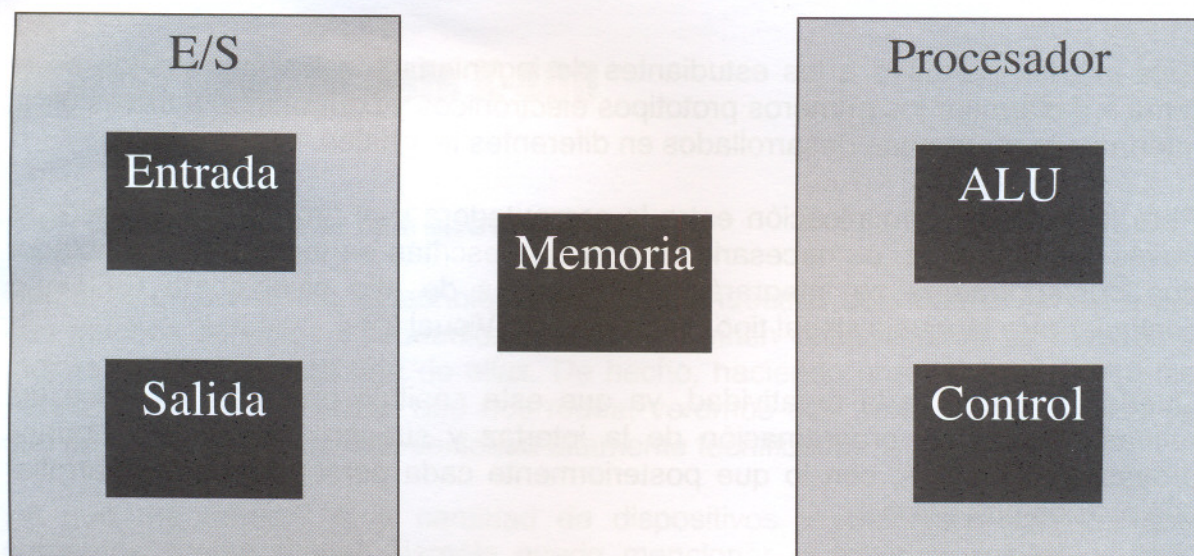
Entrada

Salida

Memoria

Procesador

ALU

Control

El procesador es el ejecutor y responsable de coordinar todas las operaciones y funciones que se llevan a cabo en la computadora, las cuales son ordenadas por el Sistema Operativo.

El bloque de memoria resulta la mejor herramienta de trabajo del microprocesador, ya que es el espacio para almacenar y manipular la información necesaria.

Y por último el bloque de entrada y salida, el cual está formado por los puertos, o puertas, por donde la información va a fluir entre el microprocesador y el mundo exterior. Los puertos están numerados, por lo que no hay posibilidad de error. Y es precisamente en este bloque donde reside el valor de la computadora, ya que este es el bloque que permite la entrada y salida de información que se estará procesando.

Uno de estos puertos, de entre 65,536 posibles, es el puerto paralelo, el cual tiene comunicación directa con el bus de datos del microprocesador.

Descripción del puerto paralelo

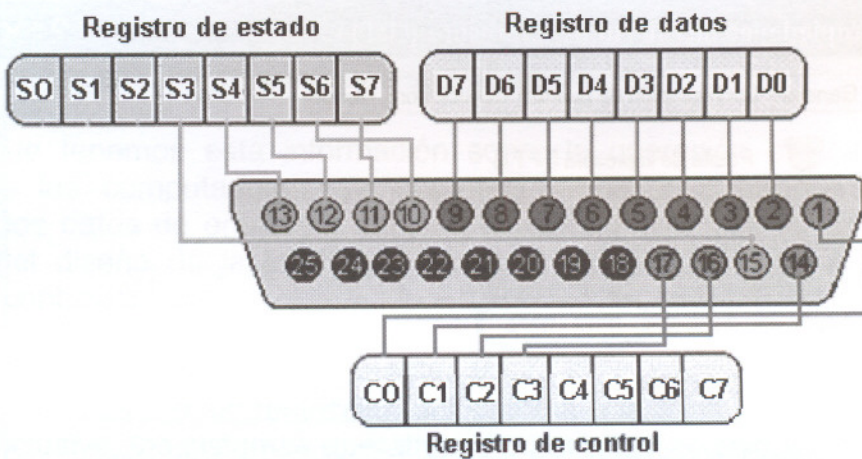
El puerto paralelo es un dispositivo de comunicación con el exterior el cual puede operar en forma bidireccional. Tradicionalmente, este puerto se le ha llamado el puerto de la impresora o puerto de impresora Centronix. Esta interfaz permite transmitir 8 bits en paralelo a través de las líneas D0 – D7 además de otras señales de control que deben estar presentes para sincronizar la comunicación entre los dos dispositivos.

Por lo tanto, este puerto de comunicación de E/S resulta ser muy útil para conectar desarrollos electrónicos propios. Lo único que tenemos que saber es la forma en la que opera, la cual se basa en tres registros de datos.

Las líneas de comunicación se pueden identificar fácilmente en el conector DB25 de la computadora y serán las responsables de coordinar la comunicación entre la computadora y el circuito periférico.

Esta comunicación bidireccional se debe dar en una coordinación de señales que permitan sincronizar el funcionamiento normal de cada parte. Así, si la computadora esta corriendo un proceso diferente al de atender el puerto paralelo, entonces podrá avisar que está ocupada y el circuito periférico tendrá que esperar. Lo mismo tendrá que suceder con el circuito si está procesando los datos y no los tiene listos para entregarlos, la computadora tendrá que esperar hasta que el circuito periférico indique que el valor en el bus de datos es válido.

Las señales del conector tipo DB25 que tienen las computadoras para comunicación del puerto paralelo se muestran en la siguiente figura:

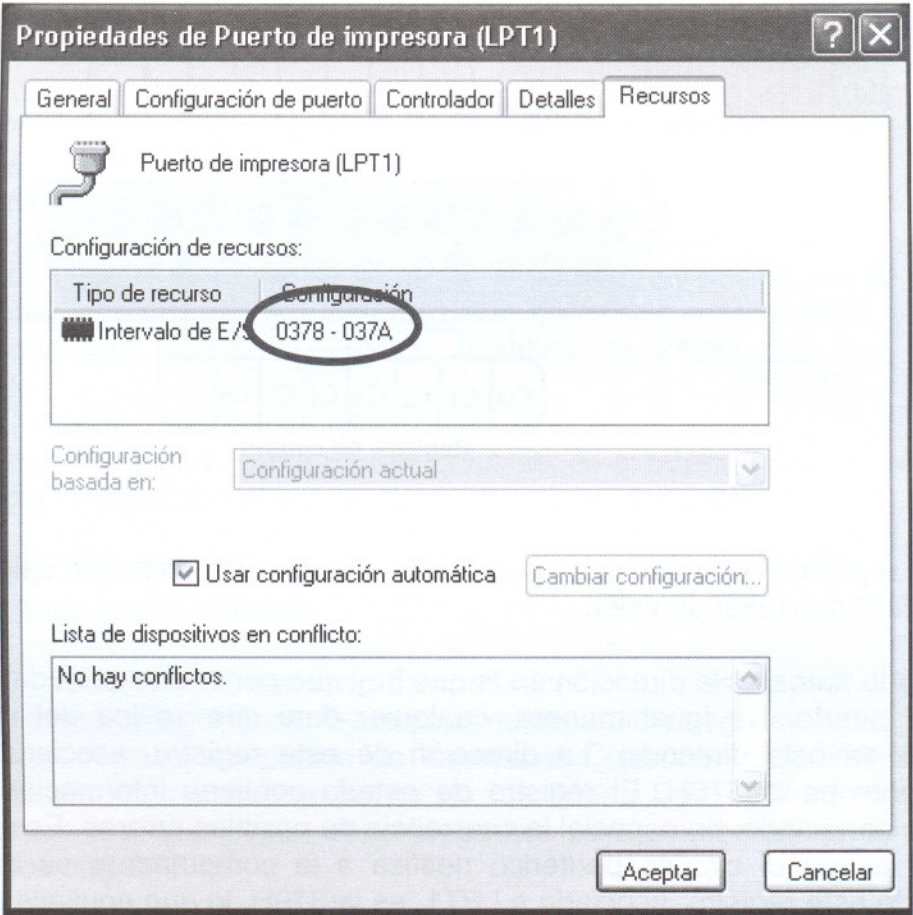
Registro de estado
S0 S1 S2 S3 S4 S5 S6 S7
Registro de datos
D7 D6 D5 D4 D3 D2 D1 D0
13 12 11 10 9 8 7 6 5 4 3 2 1
25 24 23 22 21 20 19 18 17 16 15 14
C0 C1 C2 C3 C4 C5 C6 C7
Registro de control

Estos tres registros controlan la comunicación y tienen una dirección asignada, por lo que es fácil consultar su valor.

El registro de datos es la dirección en la que hay que poner cualquier dato que sea dirigido al puerto. De igual manera, cualquier dato que se lea del exterior se encontrará en esta dirección. La dirección de este registro, asociada a LPT1, generalmente es la 378H. El registro de estado contiene información sobre el dispositivo conectado, en especial la ocurrencia de posibles errores. Es decir, es la forma en la que el circuito periférico notifica a la computadora su estado. La dirección de este registro, asociado a LPT1, es la 379H, lo que equivale al registro Base + 1. El registro de control permite inicializar el puerto paralelo de la computadora y controlar la transferencia de datos. Es la forma en la que la

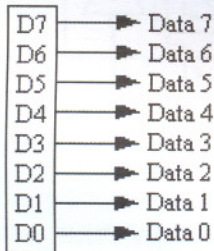
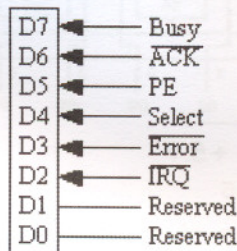
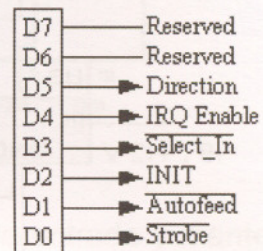
computadora comunica al periférico su estado. La dirección de este registro, asociado a LPT1, es la 37AH, lo que equivale al registro Base + 2.

Si consideráramos las direcciones de los otros puertos para impresora, la tabla sería de la siguiente forma

Puerto	Registro de Datos	Registro de Estado	Registro de Control
LPT1	378H	379H	37AH
LPT2	278H	279H	27AH
LPT3	3BCH	3BDH	3BEH

Para asegurar la dirección del puerto LPT1 de la computadora en la que vamos a trabajar, la pestaña Recursos de la Ventana de Propiedades del puerto de impresora nos indicará la dirección exacta.

Lo último que nos falta describir, es la señal que tiene cada uno de los bits de los registros de estado y control. Desde luego que el registro de datos va a tener los valores de entrada y salida.

Las señales de los registros se muestran a continuación donde es fácil ver que el registro de estado recibe información y el registro de control envía información.

Registro de datos		Registro de estado		Registro de control	
D7	→ Data 7	D7	← Busy	D7	Reserved
D6	→ Data 6	D6	← ACK	D6	Reserved
D5	→ Data 5	D5	← PE	D5	→ Direction
D4	→ Data 4	D4	← Select	D4	→ IRQ Enable
D3	→ Data 3	D3	← Error	D3	→ Select_In
D2	→ Data 2	D2	← IRQ	D2	→ INIT
D1	→ Data 1	D1	Reserved	D1	→ Autofeed
D0	→ Data 0	D0	Reserved	D0	→ Strobe

Estas señales de los registros de estado y control están codificadas para control de una impresora, cosa que en nuestro proyecto sustituiremos por las señales que se adapten al circuito del proyecto que se trabaje.

Una vez que tenemos esta información sobre la operación básica del puerto paralelo de las computadoras, ya podemos empezar a hablar sobre cómo programar los datos de entrada y salida a través de este puerto. Empezaremos hablando del diseño de la interfaz que estará en contacto con el circuito del proyecto a controlar.

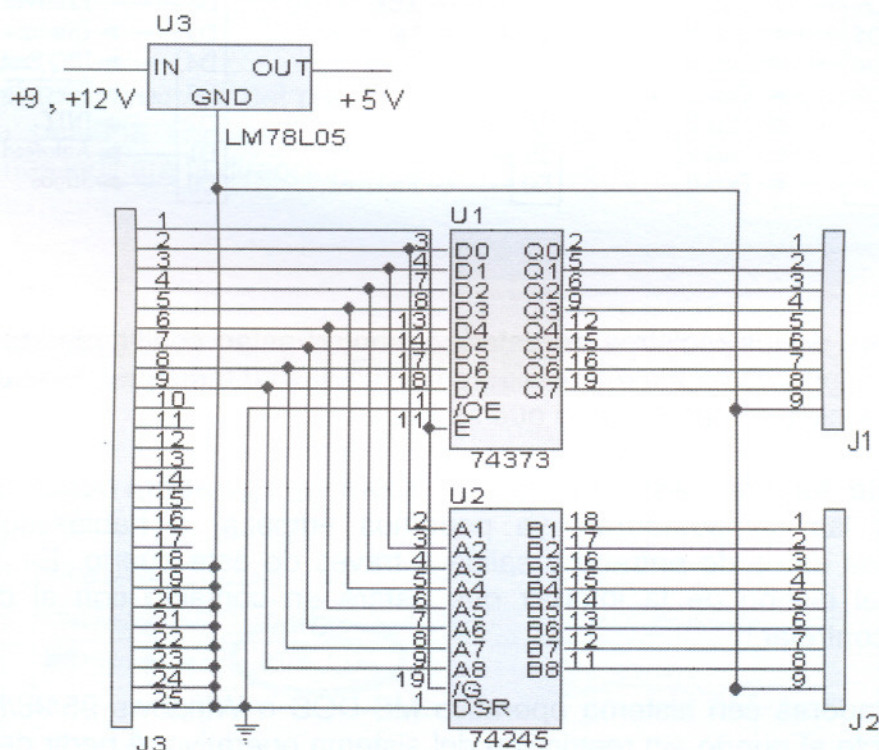
Las computadoras con sistema operativo MS-DOS o Windows 95/98/Me, tienen acceso directo al puerto sin restricción del sistema operativo. A partir de Windows NT/2000, inclusive, los sistemas operativos no permiten acceso a los puertos, por lo que en este caso se tendrá que deshabilitar esta protección.

Hay varias formas para lograrlo y en Internet se pueden encontrar aplicaciones gratuitas que permiten la configuración de un grupo de puertos o de un puerto en particular saltando las protecciones de sistema operativo.

Una aplicación de este tipo es UserPort escrito por Tomas Franzon. En el CD que acompaña el libro hay una copia de estos archivos. Es muy recomendable leer las instrucciones de instalación y siempre tener en cuenta que al usarlo, se está abriendo una protección de la computadora, por lo que hay que definir perfectamente el rango de puertos que se quieren liberar y de ser posible, volver a cerrarlos mientras no exista necesidad de lo contrario.

DISEÑO ELECTRÓNICO DE LA INTERFAZ

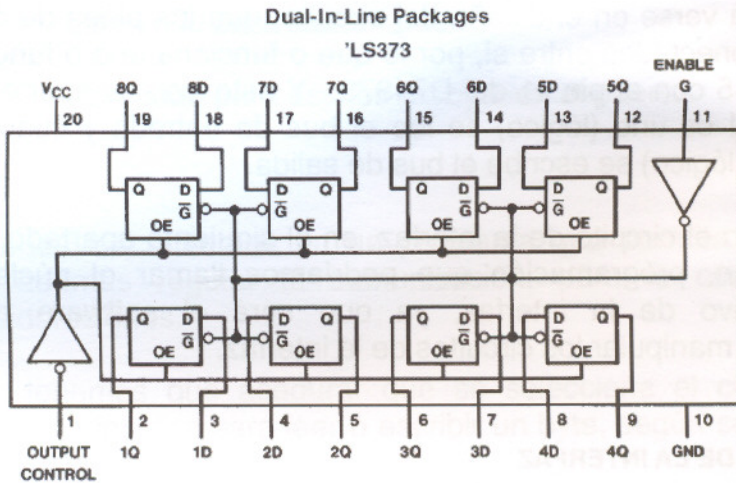
El diseño que vamos a usar es una configuración típica la cual nos va permitir dividir el puerto paralelo de la computadora en un puerto bidireccional, es decir, en un bus de entrada y uno de salida, lo cual facilitará mucho el trabajo con el circuito que vamos a controlar posteriormente.

El diagrama electrónico del diseño es el siguiente:

El diagrama muestra tres circuitos integrados y tres conectores. El conector J1 es un DB25 que se conecta a la computadora. El conector J2 manejará las señales de salida y el conector J3 manejará las señales de entrada.

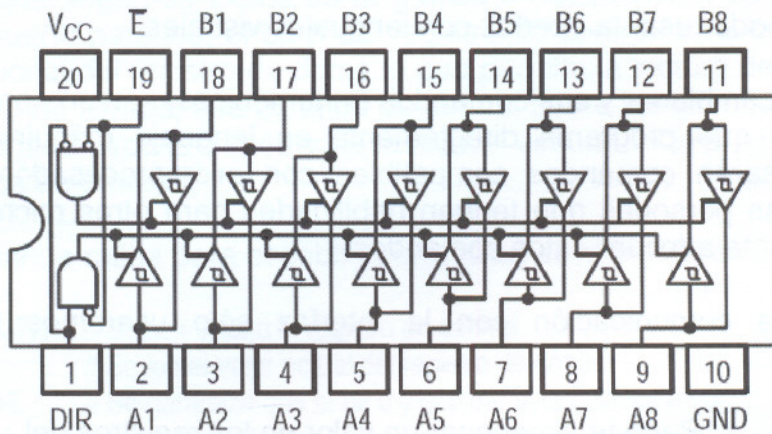
Como se usa un conector DB9, los pines 1 – 8 serán los bits datos de entrada o salida (D0-D7) y el pin 9 será tierra.

El circuito D74373 es una memoria *latch* de 8 bits. La distribución de sus pines y sus funciones se muestran a continuación:

Dual-In-Line Packages

'LS373

Una memoria *latch* va a seguir en su salida, la señal de entrada y mantendrá el valor en memoria sólo cuando se ponga un pulso en el pin de control.

Por estas características, este circuito integrado va a manejar las salidas de la computadora. Por el momento es importante hacer notar que para generar un byte de salida hay que dar un pulso en el pin de control, pulso que tendrá que ser generado por software, del que hablaremos en el siguiente apartado.

El circuito D74245 es un bus bidireccional con *buffers* de tres estados. La distribución de sus pines y sus funciones se muestran a continuación:

Por lo tanto, este circuito integrado solamente va a dejar pasar el byte de entrada a la computadora cuando se mande un pulso en el pin de control que en este caso es el pin 19 y como en el circuito anterior, el pulso que tendrá que ser generado por software, del que hablaremos en el siguiente capítulo.

Entonces, podrá verse en el diseño del circuito, que los pines de control de cada circuito están conectados entre sí, por lo que o funciona uno o funciona el otro. El pin 1 del D74245 con el pin 11 del D74373. Y esto con tal suerte que cuando el pulso de control es uno (lógico) se lee el bus de entrada y cuando el pulso de control es cero (lógico) se escribe el bus de salida.

Una vez descrito el circuito de la interfaz, en el siguiente apartado hablaremos de su programación, programación que podríamos llamar el núcleo (*kernel*) del sistema operativo de la interfaz, ya que será el software necesario para comunicarnos y manipular los circuitos de la interfaz.

PROGRAMACIÓN DE LA INTERFAZ

Ahora empezamos una parte interesante, la programación de la interfaz.

El programa o programas que permiten al usuario tener contacto con un circuito electrónico y controlarlo se le puede llamar Sistema Operativo. Así que podemos decir que vamos a iniciar la programación del sistema operativo de la interfaz.

¿Qué funciones elementales debe incluir la programación de la Interfaz? Por lo menos dos, la de lectura y la de escritura. La lectura significa leer los datos que el circuito electrónico a controlar envía a la computadora y escritura son los datos que la computadora va a enviar a dicho circuito electrónico.

Esta programación es muy conveniente que se haga en rutinas de lenguaje ensamblador, lo cual permitiría una mayor velocidad de comunicación entre la computadora y la interfaz y además, facilitaría el desarrollo de librerías dinámicas (DLL's) para poder usar la interfaz con lenguajes visuales.

El lenguaje ensamblador y sus comandos (mnemónicos) son una programación de bajo nivel, ya que programa directamente en lenguaje máquina y para este capítulo se usarán comandos compatibles con microprocesadores Intel de la familia x86. Las personas que tengan habilidades para otros microprocesadores podrán fácilmente adecuar estos comandos.

Para lograr la comunicación con la interfaz sólo usaremos las siguientes instrucciones:

MOV	Permite almacenar un valor en los registros del microprocesador de la computadora
AND	Operación booleana AND
OR	Operación booleana OR

IN Comando para leer un puerto

OUT Comando para escribir un byte en el puerto

Estas instrucciones las uniremos en un programa que permita enviar y recibir información y además regular la comunicación, para lo cual haremos las siguientes consideraciones.

1. Primero, tenemos que asegurar que se selecciona el circuito integrado correcto de la interfaz para leer o escribir un byte, según se describió en el diseño electrónico de la interfaz.

2. Después ya se podrá realizar la operación de lectura o escritura.

La rutina para leer información es la siguiente:

```
mov dx, 37A   // Se selecciona el registro de control
in al, dx     // Se lee el valor actual del registro de control
or al, 21     // Se garantiza que el bit  0 y el 5 del byte quedan en 1
out dx, al    // Se manda el valor al registro de control
mov dx, 378   // Se selecciona el registro de datos
in al, dx     // Se lee el valor del registro de datos
```

Seguramente el código y los comentarios se explican por sí solos.

Cabe aclarar que según se indica en la gráfica de distribución de señales de los registros del puerto paralelo, el bit 5 del registro de control es el que configura la forma bidireccional del registro de datos lo que permite la lectura de información, y el bit 0, la señal de *Strobe*, que es el bit de control sobre los circuitos de la interfaz y que selecciona el circuito 74245 y desactiva el circuito 74373, lo que permite la lectura de datos de entrada.

En forma similar, la rutina para escribir información es la siguiente

```
mov dx, 37A    // Se selecciona el registro de control
in al, dx      // Se lee el valor actual del registro de control
and al, DE     // Se garantiza que el bit 0 y el 5 del byte quedan en 0
out dx, al     // Se manda el valor al registro de control
mov dx, 378    // Se selecciona el registro de datos
mov al, valor  // Se carga el valor que se va a escribir en el registro de datos
out dx, al     // Se envía el valor al registro de datos
```

Como la función de escritura es inversa a la de lectura, ahora el bit 5 del registro de control es el que configura la forma unidireccional del registro de datos lo que permite la escritura de información, y el bit 0, la señal de *Strobe*, es el bit de

control sobre los circuitos de la interfaz y que desactiva el circuito 74245 y selecciona el circuito 74373, lo que permite la escritura de datos de salida y permanecerán en la memoria *latch*.

Una vez descritas las rutinas básicas en lenguaje ensamblador, ahora tenemos que hacer pequeños cambios, según lo requiera la aplicación en particular que se esté desarrollando, para que puedan funcionar con cualquier valor de salida y se pueda pasar al programa general de control de datos de entrada de la interfaz.

Para esto, podemos incluir estas rutinas en una estructura de un programa escrito en Lenguaje C, lo cual quedaría de la siguiente forma

```
#include <dos.h>

void escribir_puerto (int pto, char val);
int leer_puerto (int pto);

main ()
{
  int entrada;

  escribir_puerto(888,100);
  entrada = leer_puerto(888);

  return(0);
}

void escribir_puerto(int pto, char val)
{
  int control;

  control=pto+2;
  _asm{
        mov dx, control
        in al,dx
        and al, 0xDE
        out dx, al      // inicia puerto de escritura
        mov dx, pto
        mov al, val
        out dx, al
  }
  delay(20);
}
```

```
int leer_puerto(int pto)
{
  int dato;
  int control;

  control=pto+2;
  _asm{

        mov dx, control
        in al, dx
        or ax, 0x21
        out dx, al     // inicia puerto de lectura
        mov dx, pto
        in al,dx
        mov dato,al
    }
    delay(20);
  return(dato);
}
```

El comando **_asm{ }** permite la escritura directa en lenguaje ensamblador, respetando el formato de números hexadecimales definido por el Lenguaje C.

Así queda un programa para escribir y leer datos a cualquier número de puerto de la computadora, que en este caso el número 888 decimal equivale al número 378 hexadecimal.

PROGRAMACIÓN EN LENGUAJES VISUALES

Continuaremos con otra parte interesante, la programación de la interfaz usando lenguajes visuales.

Microsoft Visual Basic es una excelente herramienta de desarrollo con alto grado de facilidad, pero carece de algunas funcionalidades que en este momento necesitamos como es el caso del acceso directo al puerto paralelo. Pero para solventar este tipo de aparente limitación, una solución efectiva es la escritura de una librería dinámica, una DLL. Por lo tanto, primero veremos como escribir una librería dinámica usando Visual C++.

Se llaman librerías dinámicas porque su contenido no es compilado con el programa principal, sino que este solamente tiene el nombre y sus funciones y al momento de su ejecución, busca el archivo con extensión DLL donde encontrará

las funciones solicitadas. Así el programa principal de la aplicación puede ser un poco más ligero, ya que no contendrá las funciones definidas en la DLL.

Escritura de una DLL

A continuación se describen los pasos para crear una DLL usando Visual C++.

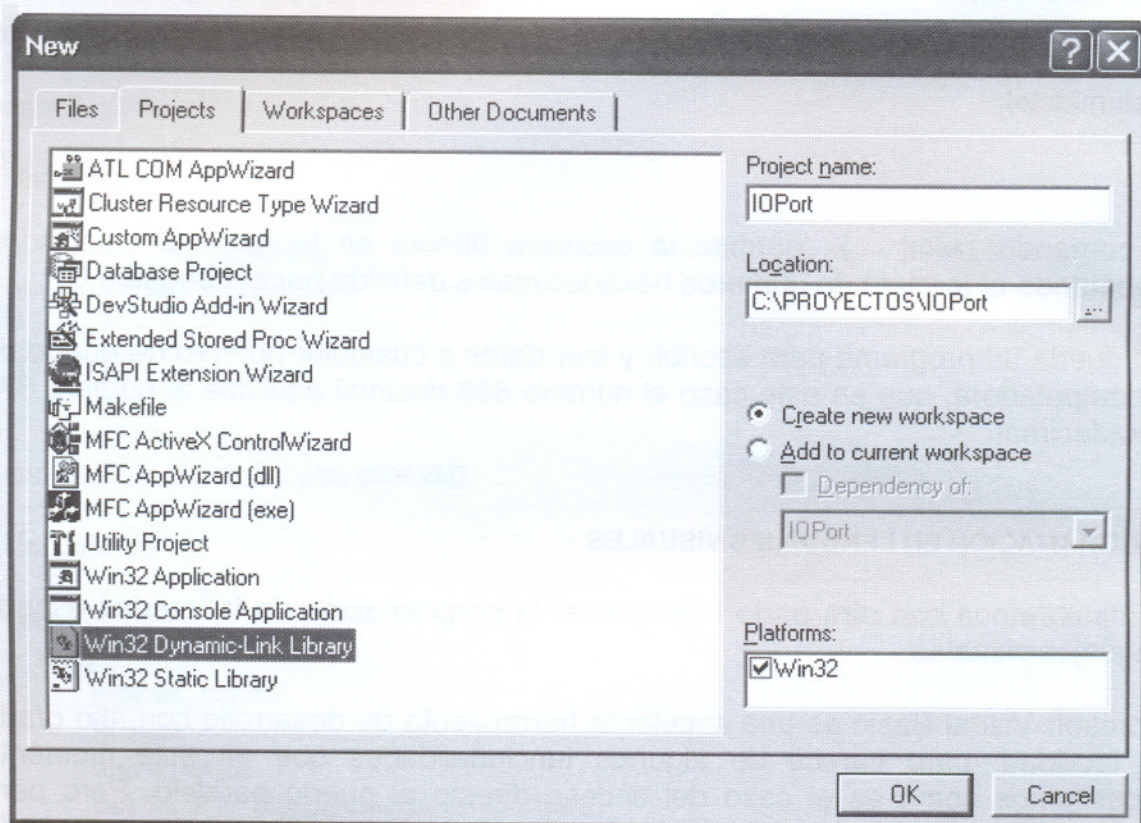
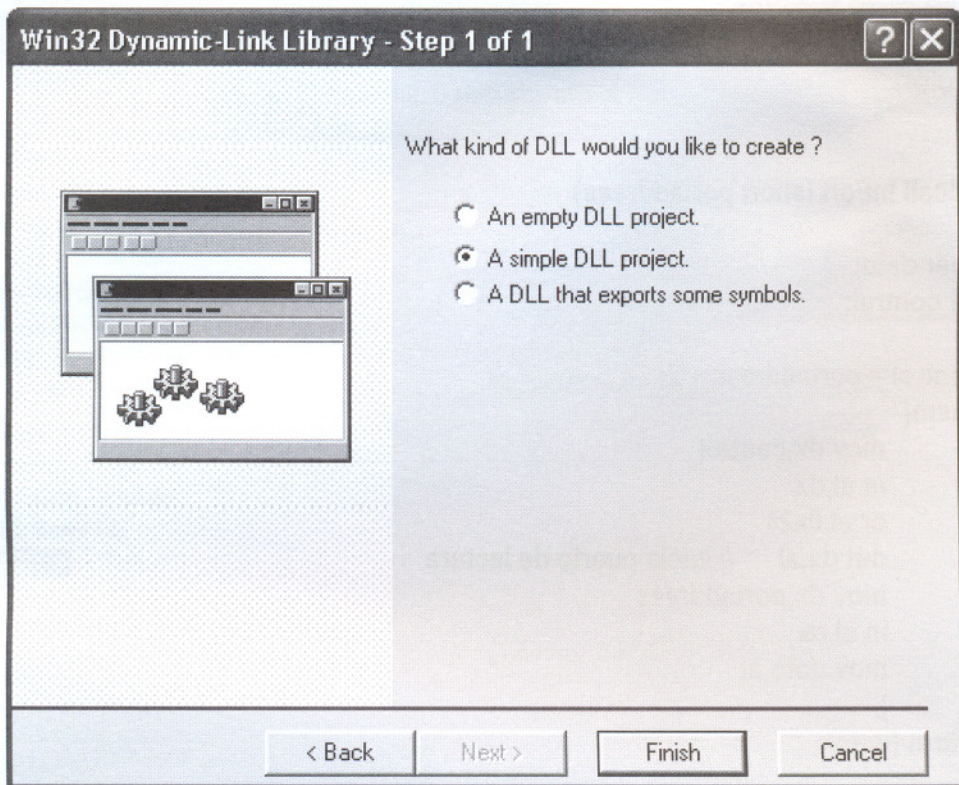
Una vez iniciado Visual C++ se selecciona **New** del menú **File**. Se selecciona **Win32 Dynamic-Link Library** de la pestaña **Projects**. Se registra el nombre del proyecto y la dirección del archivo en el que se va a grabar. Se oprime el botón **OK** y se pasa a la siguiente ventana en la que se selecciona la opción de **A simple DLL project**. A continuación se oprime el botón **Finish**.

Este proceso se ilustra en las siguientes imágenes

Win32 Dynamic-Link Library - Step 1 of 1

What kind of DLL would you like to create ?

- An empty DLL project.
- A simple DLL project.
- A DLL that exports some symbols.

< Back | Next > | Finish | Cancel

Una vez abierto el proyecto, se tendrá la siguiente presentación

IOPort - Microsoft Visual C++ - [IOPort.cpp *]

File Edit View Insert Project Build Tools Window Help

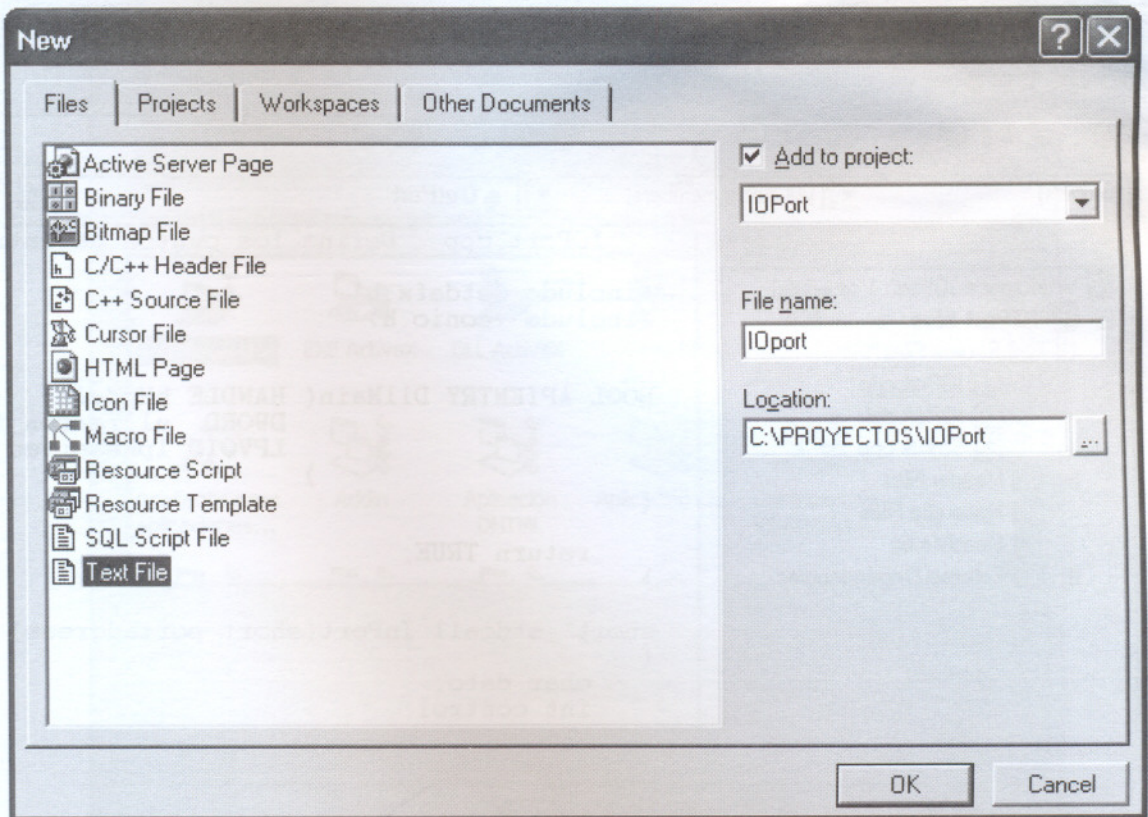
(Globals) | (All global members) | InPort

Workspace 'IOPort': 1 project(s)
- IOPort files
  - Source Files
    - IOPort.cpp
    - StdAfx.cpp
  - Header Files
  - Resource Files
  - ReadMe.txt

```
// IOPort.cpp : Define los puntos de entr

#include "stdafx.h"
#include <conio.h>


BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_f
                       LPVOID lpReserved
                     )
{

    return TRUE;
}
```

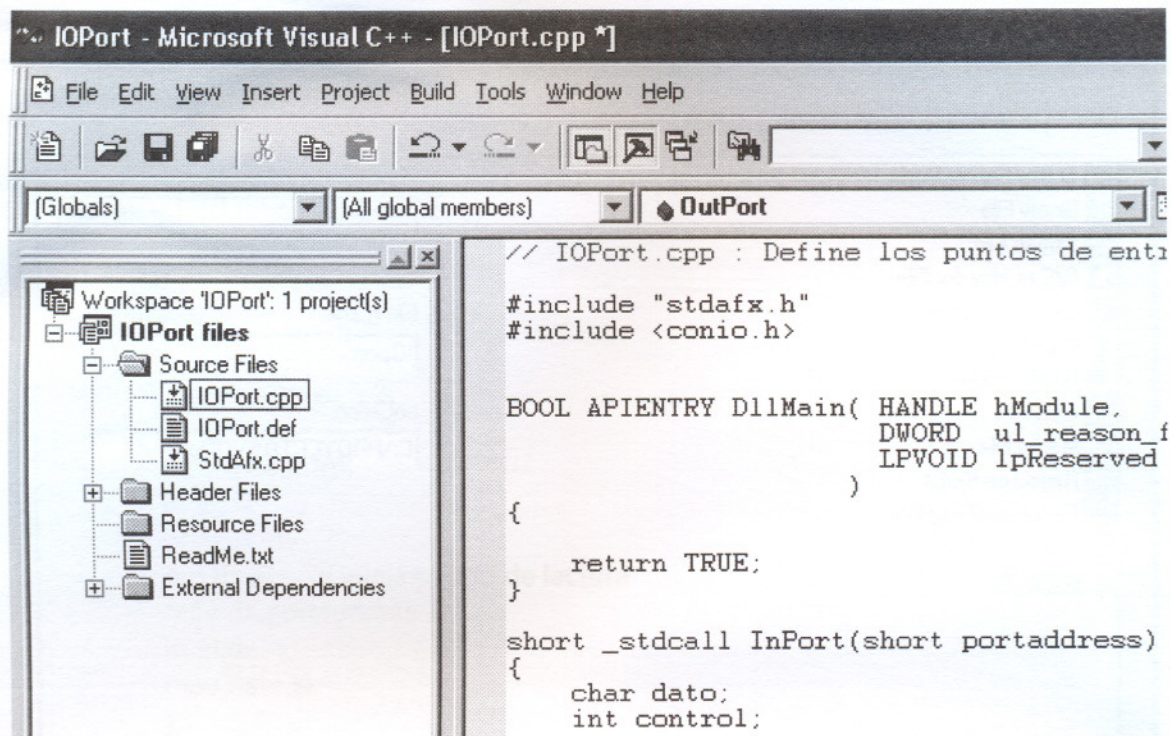
Ahora hay que agregar al final del proyecto las rutinas de comunicación escritas en Lenguaje C, según se mostró en el apartado anterior. Se muestran a continuación:

```
short _stdcall InPort (short portaddress)
{
        char dato;
        int control;

        control = portaddress + 2;
        _asm{
                mov dx,control
                in al,dx
                or al,0x21
                out dx,al       // inicia puerto de lectura
                mov dx,portaddress
                in al,dx
                mov dato,al
                }
        return (dato);
}


void _stdcall OutPort (short portaddress,char data)
{
        int control;

        control = portaddress + 2;
        _asm{
                mov dx,control
                in al,dx
                and al,0xDE
                out dx,al       //inicia puerto de escritura
                mov dx,portaddress
                mov al,data
                out dx,al
                }

}
```

El siguiente paso es crear un archivo '*.def' para poder exportar estas dos funciones. Se selecciona **New** del menú **File** y se selecciona un **Text File** de la pestaña **Files**. Se nombra igual que el proyecto.

En la ventana de trabajo del proyecto aparecerá un archivo vacío, al que hay que escribirle el siguiente código

```
LIBRARY IOPort

EXPORTS

InPort      @1
OutPort     @2
```

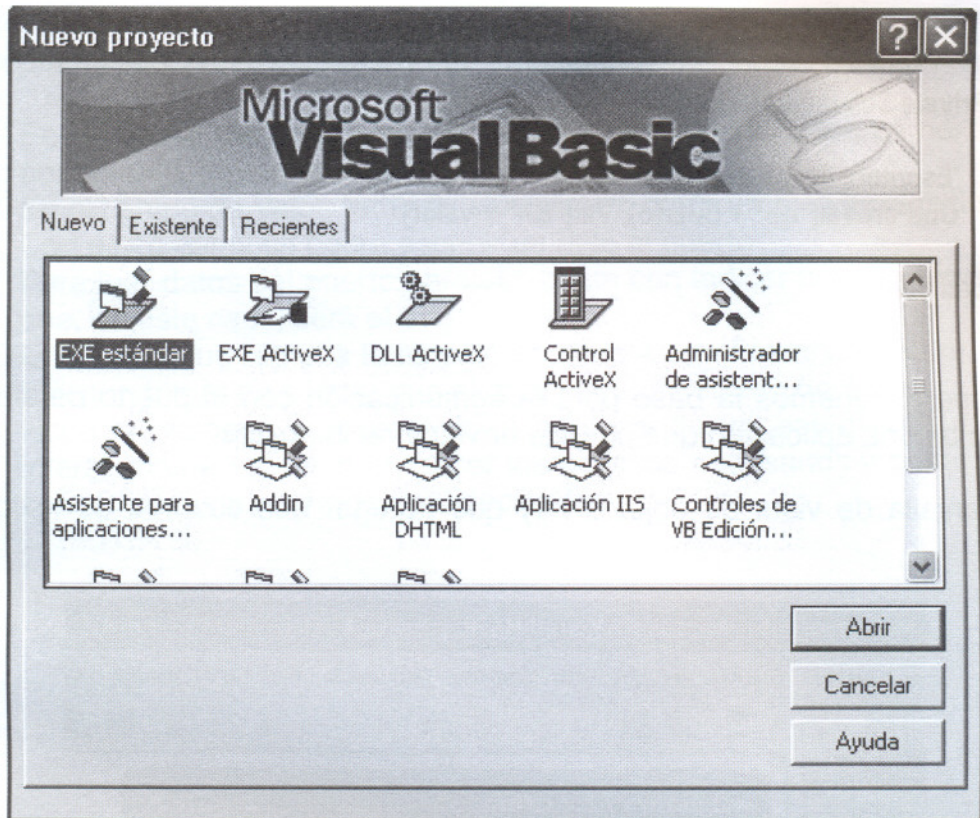
A continuación se selecciona **Build IOPort.dll** del menú **Build**. El proceso debe terminar sin errores, de lo contrario hay que revisar su causa. Una vez terminado el proceso sin errores ya se puede ver el archivo IOPort.dll en el directorio debug donde está grabado el proyecto. La ventana del proyecto se verá de la siguiente forma

```
// IOPort.cpp : Define los puntos de entr

#include "stdafx.h"
#include <conio.h>


BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_f
                       LPVOID lpReserved
                     )
{

    return TRUE;
}

short _stdcall InPort(short portaddress)
{
    char dato;
    int control;
```

En la carpeta debug aparecerá el ícono IOPort.dll

Así queda programada la librería, que ahora tendrá que copiarse en la carpeta **c:\windows\system32** para su posterior uso.

Programación de la interfaz en Visual Basic

Ya que tenemos creada la librería dinámica, ahora la vincularemos a un programa escrito en Visual Basic y con esto ya tendremos los comandos para poder tener comunicación con la interfaz.

Después de iniciar Visual Basic hay que seleccionar una aplicación estándar

En la ventana del código hay que escribir las siguientes frases, las cuales darán acceso a las funciones de entrada y salida creadas en la DLL IOPort.

```
Option Explicit
Private Declare Function InPort Lib "IOPort.dll" (ByVal portaddress As Integer) As Integer
Private Declare Sub OutPort Lib "IOPort.dll" (ByVal portaddress As Integer, ByVal data As Integer)
```

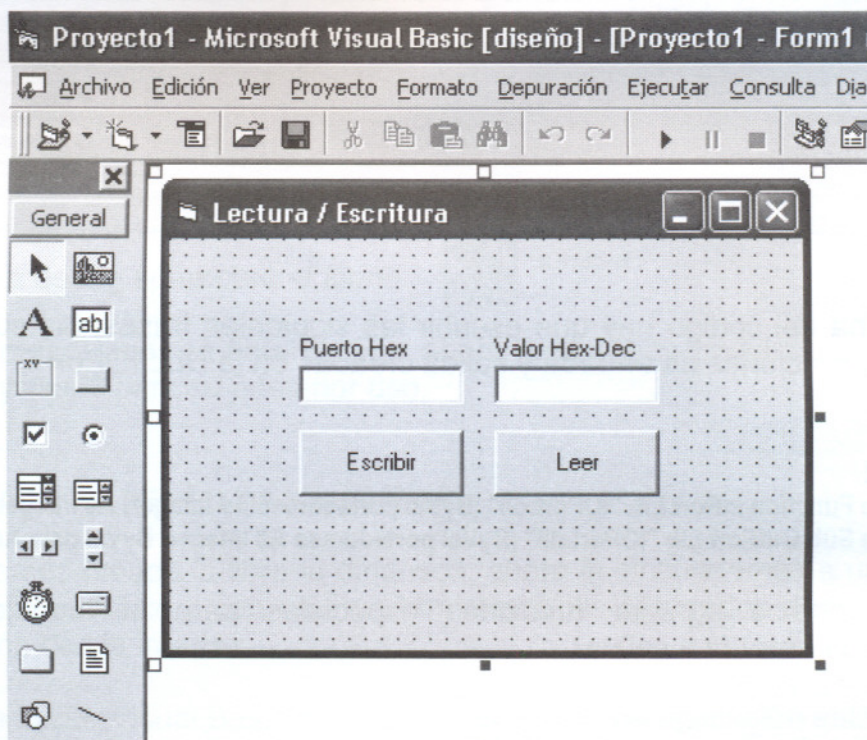
También podemos agregar funciones especiales para el manejo de esta información de entrada y salida

```
'*****************************************************************************************
'Rutinas de comunicación al puerto paralelo: LECTURA Y ESCRITURA
'*****************************************************************************************

Private Function inp(valor As String) As String

    'Lectura del puerto
    inp = InPort (Val("&h" + valor))

End Function
```

```
Private Sub out (puerto As String, valor As String)

   'Escritura del puerto
   OutPort Val("&h" + puerto), Val("&h" + valor)

End Sub
```

Ahora que ya tenemos la base para la comunicación con el puerto paralelo, hay que escribir una aplicación que permita enviar y recibir bytes.

En la pantalla de vista de objetos hay que agregar dos cuadros de texto y dos botones.

Proyecto1 - Microsoft Visual Basic [diseño] - [Proyecto1 - Form1
Archivo Edición Ver Proyecto Formato Depuración Ejecutar Consulta Dia
General
Lectura / Escritura
Puerto Hex
Valor Hex-Dec
Escribir
Leer

Después de renombrar los objetos, en la vista de código escribiremos el siguiente código

```
Private Sub cmdEscribir_Click()
   out txtPuerto.Text, txtValor.Text
End Sub

Private Sub cmdLeer_Click()
   txtValor.Text = inp(txtPuerto.Text)
End Sub
```

Al ejecutar la aplicación se deberán tener las siguientes consideraciones:

1. En el campo del Puerto se deberá de escribir la dirección en hexadecimal, por lo tanto, hay que escribir el número 378
2. En el campo de Valor, también se escribirá con números hexadecimales
3. Con estos datos se podrá escribir un byte al puerto
4. Para leer datos del puerto paralelo, basta con indicar la dirección del puerto que, en este caso, será el 378
5. El número que regresa la función aparecerá en el campo de Valor, sólo que hay que tomar en cuenta que el dato será un número decimal.
6. Hago un llamado a la creatividad personal de los estudiantes y personas interesadas en el tema en hacer los cambios necesarios y poder leer en el cuadro de texto números hexadecimales.

Capítulo 11

Integración de proyectos

Integración de proyectos

DEFINICIÓN

Una vez descubiertos todos los secretos del diseño y programación de la interfaz, ahora podemos desarrollar múltiples aplicaciones que nos ayudarán a unir, como se comentaba al principio, el mundo de la electrónica con el mundo de la programación y el control.

A continuación se ofrece una lista de proyectos, ya desarrollados, con los que se podrán adquirir experiencia en el uso y diseño de aplicaciones mecatrónicas usando la interfaz de puerto paralelo o de cualquier otro tipo de formato de comunicación, ya que los diseños electrónicos no son sensibles a este formato.

En el CD del libro se encuentra la codificación sugerida para el control de los diferentes proyectos, combinando entre soluciones programadas en lenguaje C y otras en lenguajes del Microsoft Visual Studio (Visual Basic y Visual C++).

También se podrán adquirir los proyectos propuestos en forma de productos comerciales o *KITS*, en los que se incluyen todos los componentes electrónicos necesarios, una tarjeta de circuito impreso que permitirá una fácil manipulación y operación del diseño electrónico y un instructivo de armado.

RECOMENDACIONES PARA ENSAMBLAR Y SOLDAR LOS KIT DE PROYECTOS

Para ensamblar con éxito los componentes electrónicos en las tarjetas de circuito impreso hay que tomar en cuenta las siguientes recomendaciones, que describen los procedimientos para preparar, soldar y probar los kits de proyectos. Con las herramientas adecuadas y un poco de paciencia se podrá llevar con éxito este trabajo.

1 - Herramientas básicas:

- Cautín tipo lápiz de 30 watts
- Pequeño trapo de tela para limpieza e
- Pinza de punta pequeña
- Pinza de corte
- Desarmador de relojero
- Soldadura de estaño, 1mm 60/40, con alma de resina.

2 – Garantizar que se cuenta con todos los componentes indicados en el instructivo de cada kit.

Una vez cumplidos los pasos anteriores, se procederá al armado, para lo cual se seguirán los siguientes pasos:

1. Conecte el cautín a la corriente alterna para que inicie su calentamiento

2. Ubique la distribución de los componentes tomando como referencia la guía de componentes que se presenta en la placa de circuito impreso y en los instructivos de cada kit.

3. Doble los terminales de los componentes (cuando sea necesario) para lograr una correcta inserción en la placa de circuito impreso.

4. Inserte los componentes siguiendo el orden, posición y sentido indicado en los instructivos de cada kit.

5. Compruebe que el cautín tiene la temperatura correcta. Una forma de comprobarlo es tocar la punta con la soldadura de estaño; si la soldadura se derrite inmediatamente, entonces la temperatura es correcta. Limpie la punta del cautín con el trapo de tela previamente humedecido con agua y ya está listo para soldar.

6. Caliente la unión del componente con la placa de circuito impreso solo un par de segundos, ya que el exceso de temperatura puede dañar la placa de circuito impreso y/o el componente.

7. Toque la punta del cautín con el extremo de la soldadura de estaño sin retirar el cautín de su posición actual. Cuando el estaño se derrita sobre la placa de circuito impreso, retire inmediatamente la soldadura de estaño y el cautín de la placa de circuito impreso. Deje enfriar el punto soldado varios segundos.

8. Corte el excedente de la terminal del componente en caso de ser necesario.

9. Por último revise que la soldadura de estaño haya quedado en forma uniforme en el punto soldado. Una buena soldadura debería verse como un pequeño cono de estaño rodeando completamente la unión de la terminal del componente con la placa de circuito impreso.

10. El acabado de la soldadura debe tener un aspecto brillante, de no ser así, significa que se ha realizado una soldadura fría. El peligro de la soldadura fría es el hecho de que pueden quebrarse y realizar falsos contactos en un futuro.

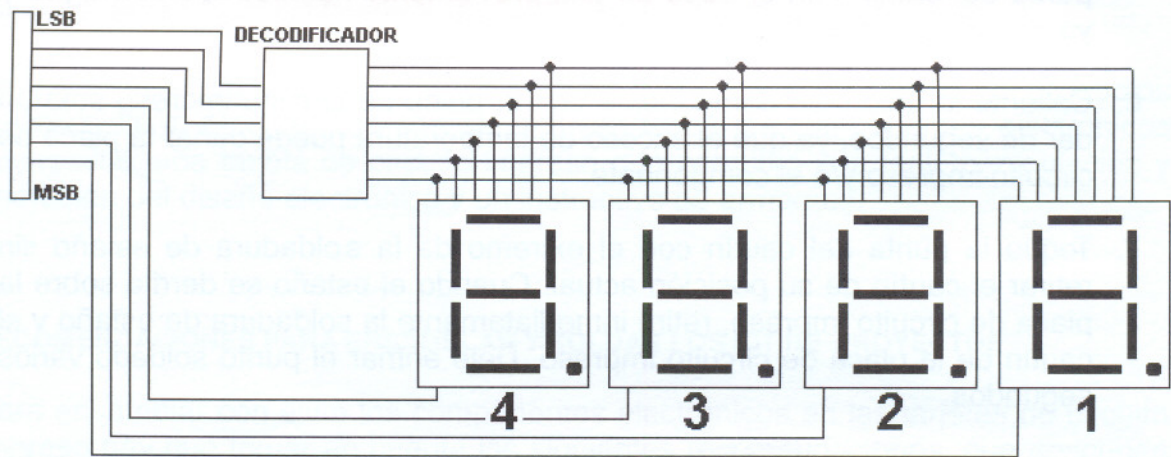
11. Puede ser normal que alrededor del punto soldado quede una pequeña mancha amarilla. Esto es el resto de la resina que trae la soldadura de

estaño en su interior. Para removerla raspe suavemente la mancha con el desamador de relojero.

CONTROLADOR DE UN DISPLAY DE 4 DÍGITOS

Es divertido mostrar los números en un display de 7 segmentos, pero es más divertido programar cuatro dígitos por medio de la computadora, lo que nos permitiría manejar cantidades desde 0 hasta 9999.

Este proyecto permite controlar perfectamente cuatro *displays* de siete segmentos usando un solo decodificador y simulando por software el multiplexor, el cual será el componente que seleccione el dígito con el número indicado en el codificador.

El diseño electrónico consiste en cuatro *displays* de 7 segmentos, un decodificador, cuatro transistores los cuales son usados como interruptores electrónicos para prender/apagar cada dígito y elementos pasivos de polarización.

El algoritmo se puede describir de la siguiente forma:

Con un bus de datos de 8 bits se pueden prender hasta cuatro dígitos, ya que los cuatro bits menos significativos pueden indicar el número deseado a través del decodificador, y los cuatro bits más significativos seleccionarán, en forma directa, el dígito deseado.

De esta forma, para prender algún dígito basta con mandar el número deseado al decodificador y el bit de selección del dígito. Por ejemplo, para prender el segundo dígito con el número cuatro, el byte de datos debería ser 0010 0100. Para prender los cuatro dígitos con el número tres, el byte de datos sería 1111 0011.

Una vez aclarados estos conceptos, ahora mediante el uso de la programación es posible presentar el número que se quiere.

Para mostrar el número 1234, se tendrá que programar la salida de la siguiente secuencia de bytes, considerando que si la velocidad en la repetición del ciclo no es suficiente, esto podría causar un parpadeo de los dígitos.

1000 0001
0100 0010
0010 0011
0001 0100

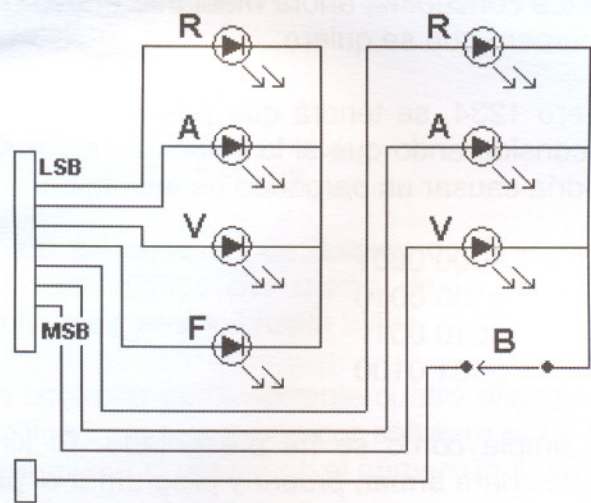
Esto puede ser tan simple como se ha presentado. El kit K-405 contiene los componentes necesarios para armar, probar y programar el proyecto. En el CD del libro hay varios ejercicios de programación codificados en lenguaje C y también en lenguaje ensamblador para microprocesadores Intel de la familia x86.

CONTROL Y PROGRAMACIÓN DE UN SEMÁFORO

Un semáforo es un dispositivo muy común en las calles de cualquier ciudad. Pero al programarlo, nos daremos cuenta que no es un proceso tan obvio. El proyecto se puede complicar un poco si queremos programar un crucero que incluya por lo menos dos sentidos, señales de vuelta y un botón para que los peatones puedan detener la circulación y cruzar al otro lado.

El kit K-410 contiene los componentes necesarios para armar, probar y programar una simulación de este proyecto. En el CD del libro hay varios ejercicios escritos en lenguaje C que permiten simular las siguientes operaciones:

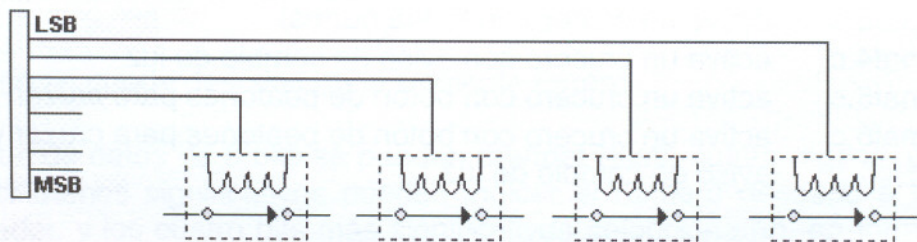
- semaf1.c activa un semáforo en forma normal
- semaf2.c activa un semáforo con aviso de cambio de luz
- semaf3.c activa un crucero en forma normal
- semaf4.c activa un crucero con aviso de cambio de luz
- semaf5.c activa un crucero con botón de peatones para cruzar
- semaf6.c activa un crucero con botón de peatones para cruzar y con aviso de cambio de luz

R
A
LSB
V
MSB
F
R
A
V
B

CONTROL DE RELEVADORES

Las computadoras manejan electrónica digital de baja potencia. Para conectar y controlar circuitos de mayor potencia, los relevadores son una buena opción. Los relevadores son interruptores electro-magnéticos que, con poco voltaje y corriente, son activados y a través de sus contactos mecánicos podemos hacer pasar mucha más potencia de la que maneja una computadora digital.

Este proyecto permite controlar perfectamente cuatro relevadores usando cuatro bits de control, uno para cada relevador. El diseño electrónico consiste en cuatro relevadores, cuatro transistores los cuales son usados como interruptores electrónicos para activar/desactivar cada relevador y elementos pasivos de polarización.

El algoritmo se puede describir de la siguiente forma:

LSB
MSB

De esta forma, para activar algún relevador basta con mandar el bit deseado a la interfaz. Por ejemplo, para activar el tercer relevador, el byte de datos debería ser 0000 0100. Para activar los cuatro relevadores, el byte de datos sería 0000 1111.

Una vez aclarados estos conceptos, ahora mediante el uso de la programación es posible activar la secuencia deseada de relevadores, o programar diferentes tiempos de activación.

El kit K-415 contiene los componentes necesarios para armar, probar y programar el proyecto. En el CD del libro se presenta un ejercicio programado en un lenguaje visual (MS Visual Basic) que permite controlar los relevadores.

ADQUISICIÓN DE DATOS A TRAVÉS DE UN CONVERSOR ANALÓ GICO-DIGITAL

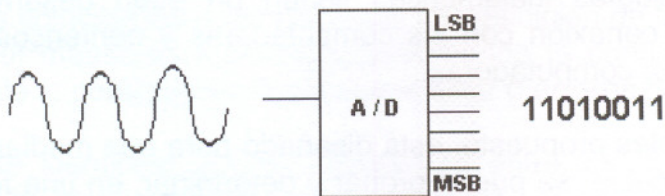
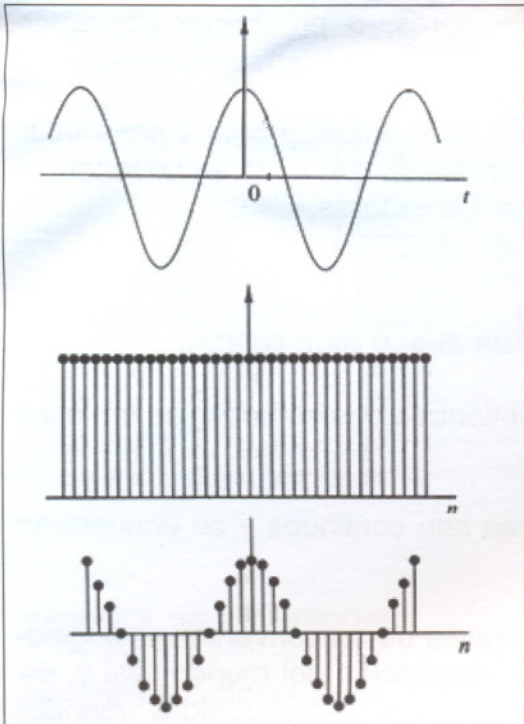
El mundo de las computadoras es digital, sólo entienden dos estados, los famosos unos y ceros.

El mundo real es analógico, es decir, sus valores son continuos y se encuentran en diferentes rangos de valores.

La forma de comunicar estos dos mundos es a través de un conversor analógico-digital, el cual permitirá a la computadora tener información del mundo real y, de un conversor digital-analógico, lo que permitirá a la computadora mandar información al exterior.

El kit K-420 contiene los componentes necesarios para armar, probar y programar este proyecto que abre las posibilidades de adquisición de datos de dispositivos de la vida real.

La electrónica de este proyecto es simple en cuanto al número de componentes, ya que sólo contiene un circuito integrado ADC y un oscilador hecho con una resistencia y un condensador.

LSB
A/D
11010011
MSB

Según el número de bits de salida, será la capacidad del conversor, ya que a mayor número de bits, la resolución y el rango de amplitud de la señal a manejar serán mayores. En caso de usar 4 bits, solamente se podrán distinguir 16 valores en el rango total de amplitud de la señal analógica.

La operación de los conversores analógico-digitales se basa en dos características, que son el circuito de muestreo de la señal y el módulo de conversión.

En la parte superior de la figura que se muestra a la izquierda, se ve la señal que se quiere convertir. En la parte central se representa el circuito de muestreo que tomará una muestra de la señal con una frecuencia fija. En la parte inferior se ven los valores recolectados por el circuito de muestreo que simula la señal original, incluyendo solamente los valores muestreados.

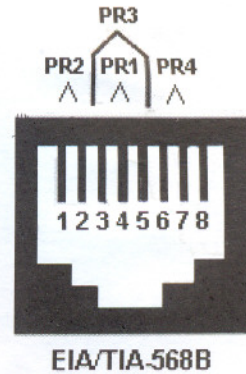
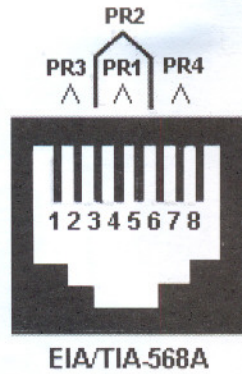
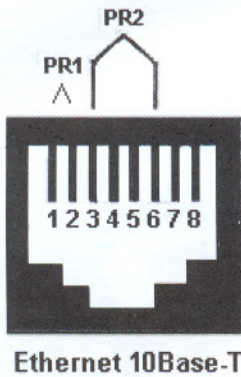
PROBADOR DE CABLES DE RED

Las redes de computadoras cada vez ofrecen más servicios y beneficios, por lo que su uso se ha ampliado en las empresas así como en los hogares. En los últimos 25 años su desarrollo ha llegado a tal punto que ahora se denominan redes convergentes, es decir, a través del mismo cableado de red se transmiten datos, voz y video.

Aunque las tecnologías inalámbricas tienen un buen desarrollo, el cableado estructurado y de conexión con las computadoras y periféricos, sigue siendo el pilar de las redes de computadoras.

El probador de cables propuesto, está diseñado para que mediante el escaneo de las terminales del cable, se pueda probar y determinar, en una forma muy rápida, la continuidad, desconexión y/o correcta polarización de los pares trenzados del cable. En el ejercicio propuesto hasta el momento, se ha desarrollado el algoritmo para probar los siguientes estándares:

- Ethernet 10Base-T
- EIA/TIA-568A, EIA/TIA-568B
- Cable Cruzado

Ethernet 10Base-T | EIA/TIA-568A | EIA/TIA-568B

Este algoritmo se basa en criterios comunes para el reconocimiento de la continuidad entre los dos extremos del cable y posteriormente la determinación exacta de su configuración o, en su defecto, la detección de fallas de parcheo. En forma descriptiva se puede enunciar de la siguiente forma:

1. Determinar la continuidad entre las terminales, lo que indicará una posibilidad de configuración. Si se detecta continuidad entre todas las terminales del conector, se puede descartar la posibilidad de una configuración Ethernet 10Base-T y suponer alguna de las otras dos configuraciones posibles, EIA/TIA 568A/B, o de cable cruzado.

2. Si la continuidad se registra entre todas las terminales, se analiza la polaridad de las terminales necesarias para distinguir si el cable es cruzado o no.

3. Una vez confirmada alguna posible configuración, se analiza la continuidad y polaridad de cada una de las terminales del conector para garantizar la correcta configuración y parcheo del cable.

4. En caso de coincidir las mediciones con la definición de la configuración, se despliega el mensaje en pantalla confirmando la configuración detectada del cable, de lo contrario se despliega el error o errores detectados.

5. En caso de identificar alguna continuidad entre los conectores, pero no identificar la configuración, se desplegará el mensaje de cable no reconocido.

El kit K-425 contiene los componentes necesarios para armar, probar y programar este proyecto que abre las posibilidades de medición de cables con configuraciones muy diversas. Este kit incluye un CD con una aplicación visual que muestra la identificación de las configuraciones propuestas.